Graded unit brief 2  
Development phase

*Cameron Morrison 569530*

Contents

[1 Coding of the problem domain 1](#_Toc70084990)

[2 Coding of the UI domain 6](#_Toc70084991)

[3 Use of unfamiliar libraries and/or constructs. 16](#_Toc70084992)

[4 Error handling 18](#_Toc70084993)

[5 Internal documentation 24](#_Toc70084994)

[5. Test plan 25](#_Toc70084995)

[6. Test Data 29](#_Toc70084996)

[7 Test runs 35](#_Toc70084997)

[8. User documentation 45](#_Toc70084998)

[9. Bibliography 48](#_Toc70084999)

**Figures**

[Figure 1 Taken from the oracle install guide. 43](#_Toc70015628)

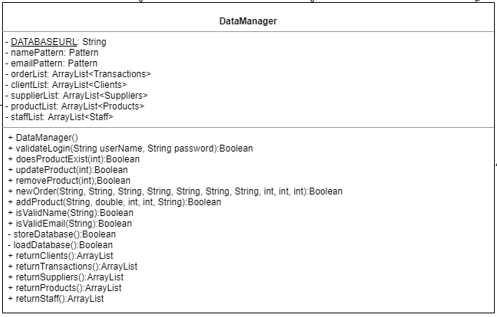
[Figure 2 Taken from the oracle install guide. 43](#_Toc70015629)

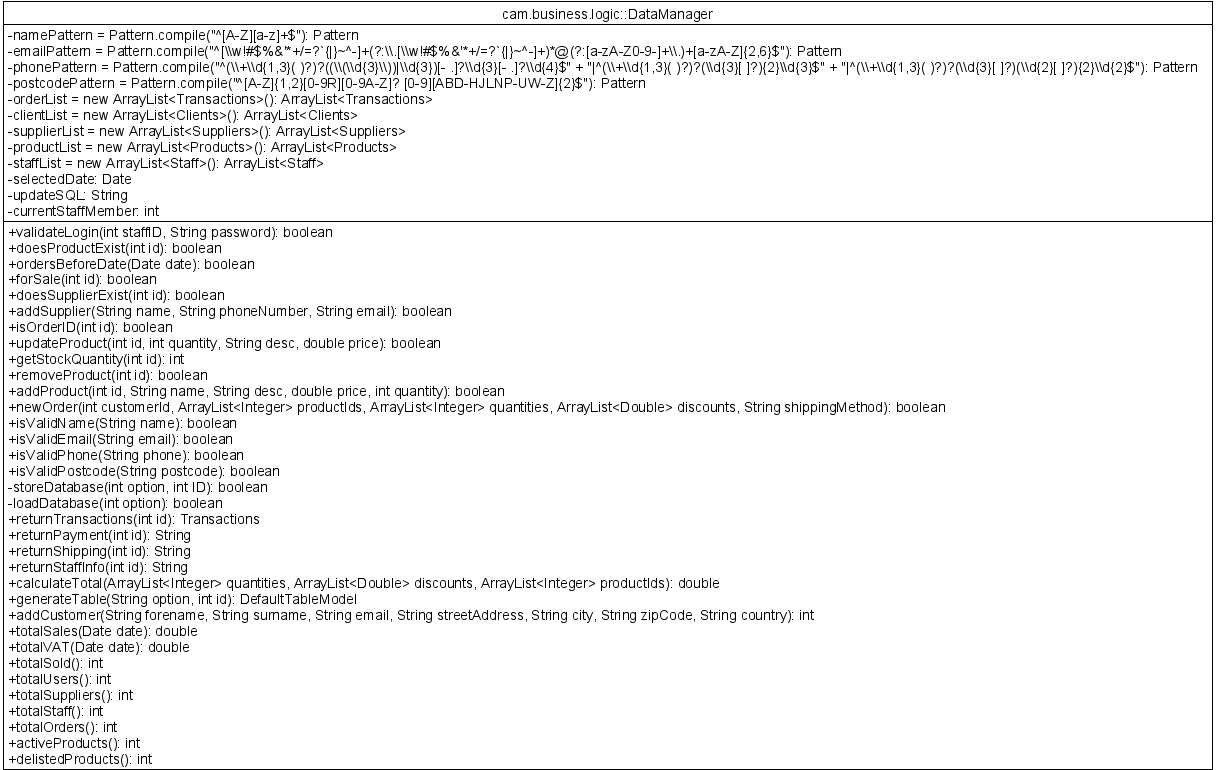
# 1 Coding of the problem domain

Whilst the design of the program ended up being closely modelled on the initial design, there were quite a few improvements. The program ended up being more complex than initially presumed. The DataManager class ended up being larger than anticipated. The DataManager class is largest class in the application and operates all the business logic.

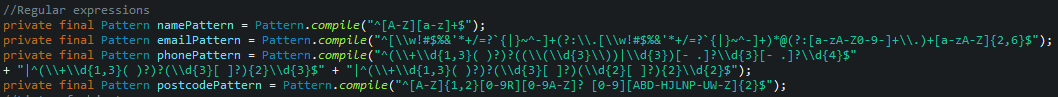
All classes in the business logic are protected except dataManger and transactions, meaning they can only be accessed by classes in the same package.

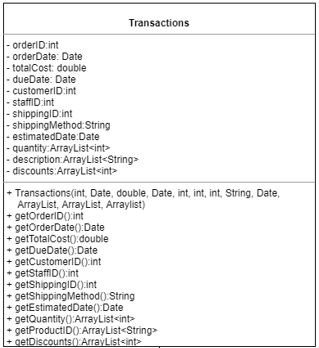
**Initial DataManager class design**

  
**Implementation of DataManager class design**



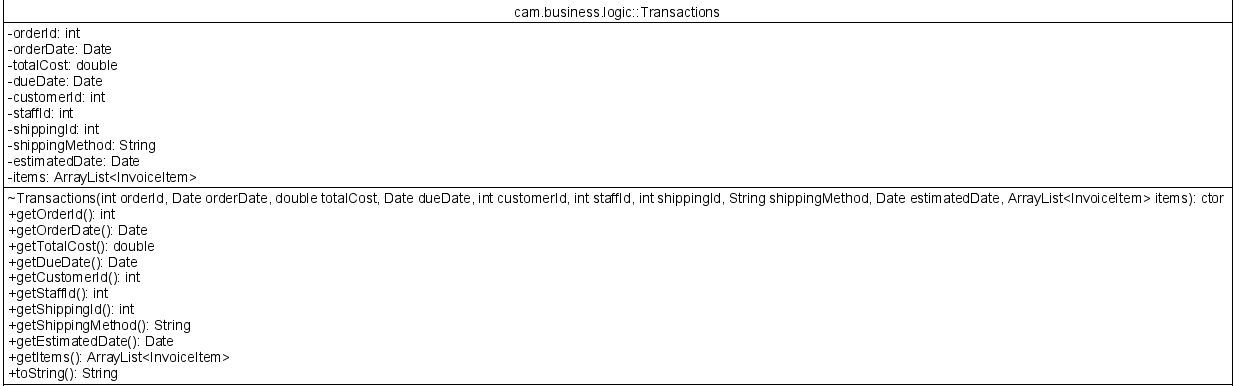
Whilst the basic plan of storing objects in private array lists stayed the same and the limited access to databases stayed just as secure as the design, extra methods were added to provide the user interface with further functionality.   
I strongly believe that my final design held true to security and data integrity as the system is not populated with data until the correct password has been entered. The password as discussed was stored as a hash in the database and not in the code. The only way to login successfully is to enter a password that when hashed is identical and related to the corresponding staff Id.   
  
My code remained highly modular, and many methods were used vigorously. For many methods I incorporated switch statements where necessary to avoid creating whole other method. By removing the concept of returning entire array lists from the business logic to the user interface package, I managed to save a substantial amount of computational power that could potentially slow down the program on older devices. By populating the tables and returning them in generateTable() I managed to keep the code reusable and safe from manipulation.

Many of the new methods were to return specific information from the other classes in the business logic package, such as clients, transactions, and products, which are used to populate the user interface.  I also have four regular expression pattern instance variables to validate user inputs format. They are accessed by four Boolean methods that return whether the string complies to the expression’s layout. In the design I only had a pattern for name and email, but I have also implemented an international phone pattern and a United Kingdom postcode pattern.

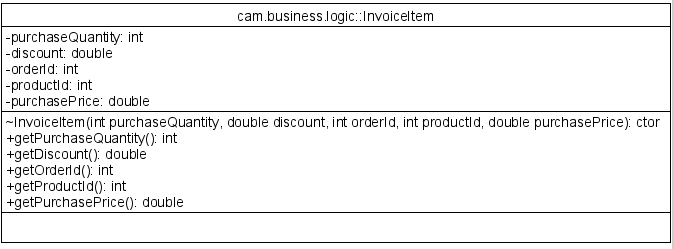
**Transactions class**

**Original design**

From my original design of the transactions class, I have removed the array list of descriptions, quantity and discounts and replaced it with a singular array list of invoice item objects. My transactions class stores the information for the order and shipping tables.

**Implementation of transaction class**  
****

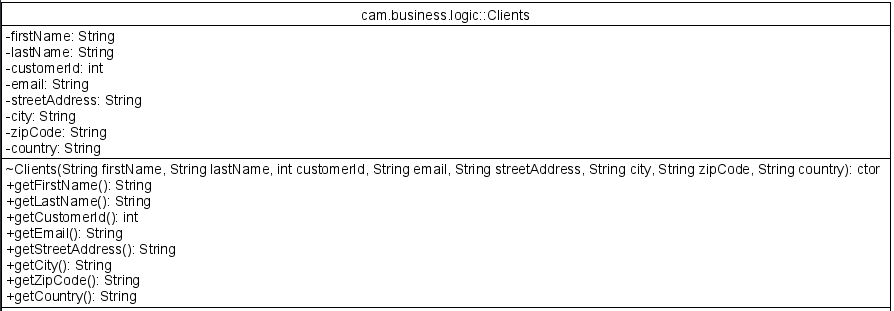
**Invoice item Implementation**

****

Whilst this was not a class on my initial class diagram. I decided it would be magnitudes easier storing the invoice item table as its own class. The list of objects is then referenced in the transactions class. That way just like the entity relationship diagram, order can have one or more invoice items. Those invoice items are unique and grouped together with a single transaction. Invoice item class allows me to store the product number, the discount, the price at the time of the sale and the related order id.

Invoice item class shares an aggregation relationship with DataManager as the other classes do. DataManager is not reliant on having an invoice item but it may have many if an order is created.

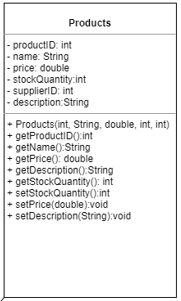
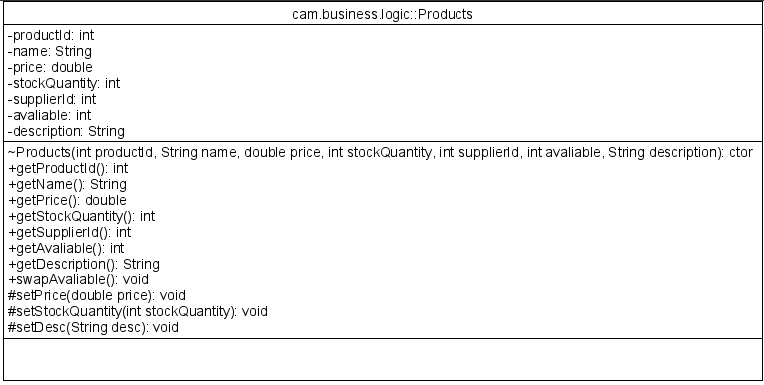
**Clients class**

**Design Implementation**

The design and implementation of the client’s class remained identical.

**Products class**

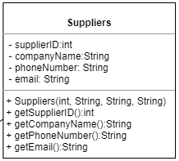
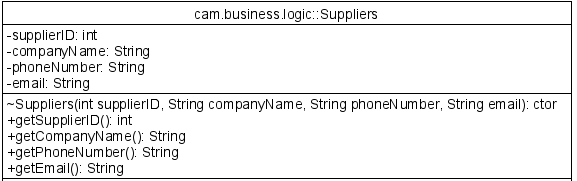
**Design Implementation**



I modified products to include avaliable, an integer value that stores 0 or 1. This variable indicates whether the product is no longer available. I have had to create a getter and setter for the instance variable.

**Suppliers class**

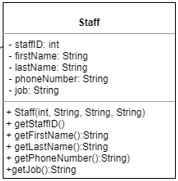
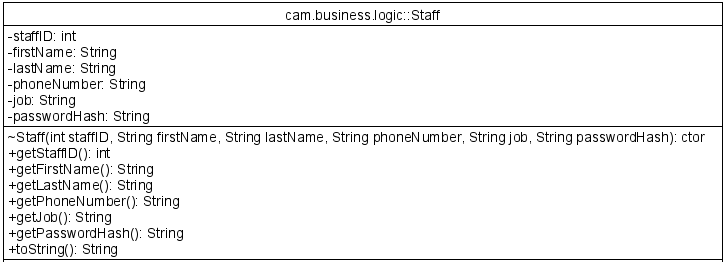
**Design Implementation**

****

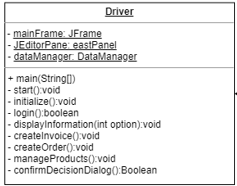
Suppliers class remained identical.

**Staff class**

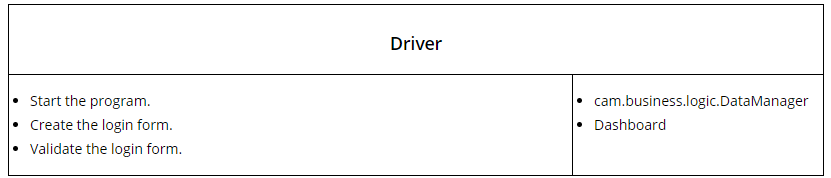
**Design Implementation**

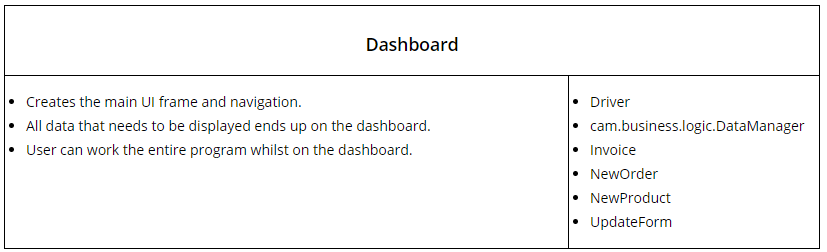
The Staff class remained similar, however I added a password hash string. I use the password hash getter to compare the users login password. The entered password is hashed and compared to the password stored in the staff class.

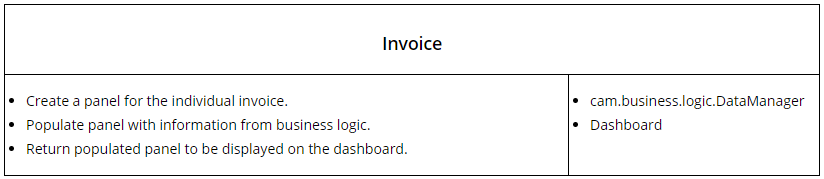
# 2 Coding of the UI domain

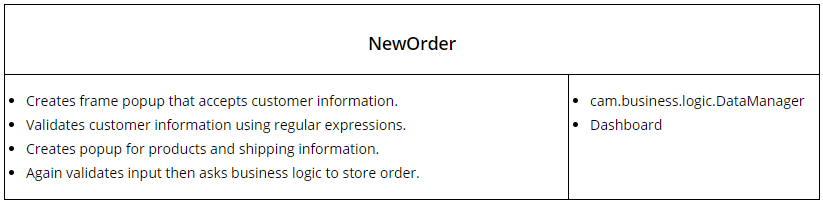
In my design I grossly oversimplified the complexity of user interface. To make the implementation easier I ended up making my interface more modular. I have six classes including the driver and used eclipse windows builder to design the basic structure with group layouts.   
I have created some class responsibility collaboration (CRC) cards to demonstrate the new classes I have created and their purpose.

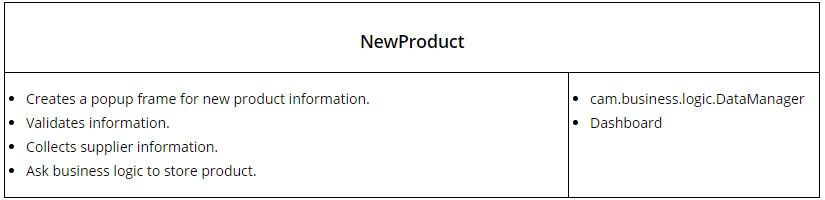
The CRC cards explain the purpose of the class on the left and the connections to other classes on the right.

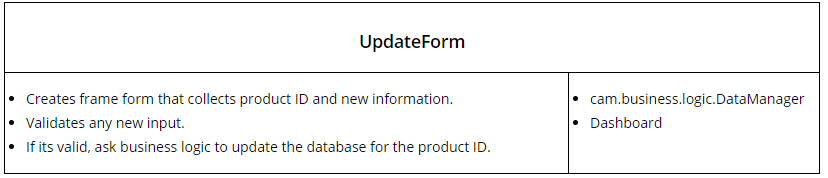




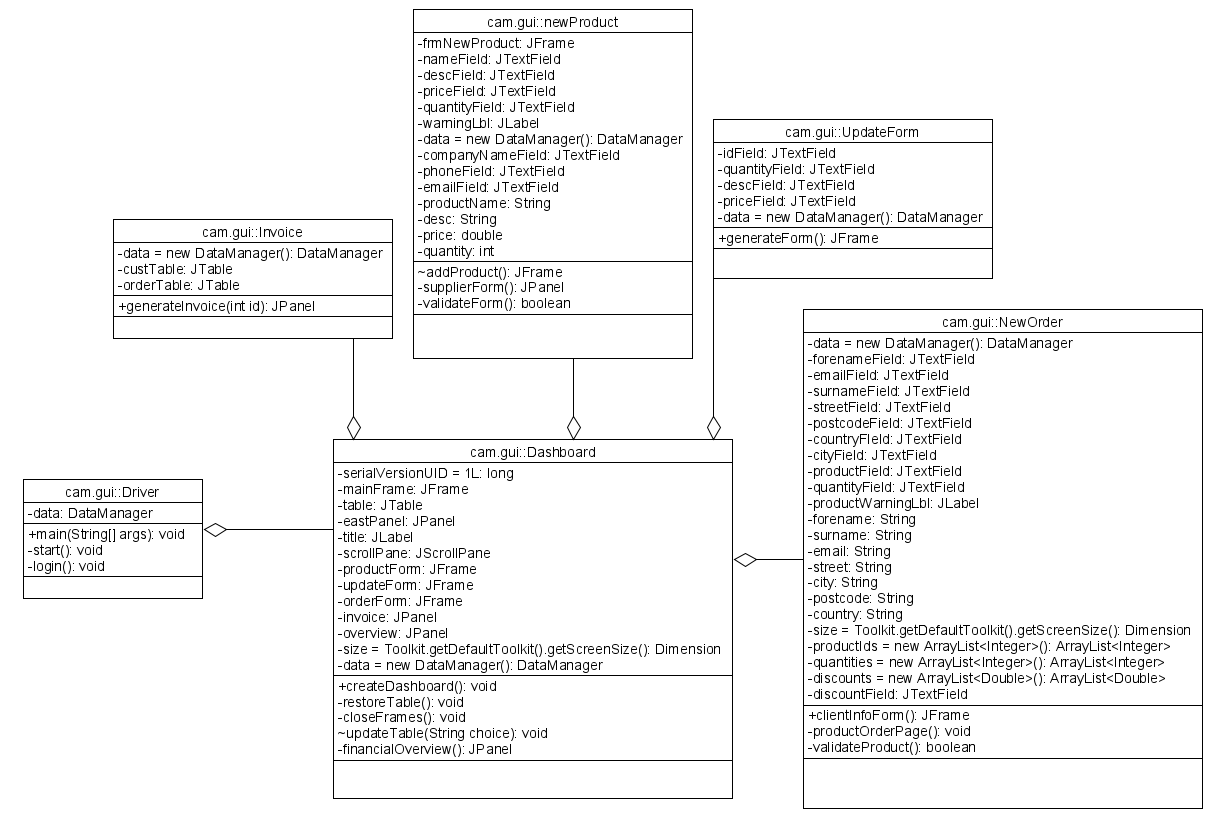




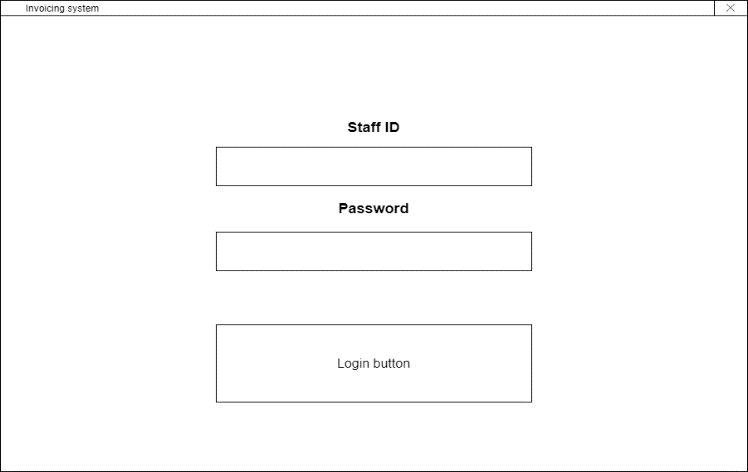
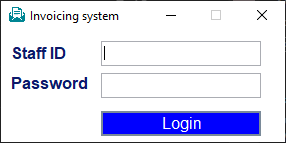




Here is a static diagram for the user interface package. Each of these classes has a relationship with the DataManager class in the business logic package.

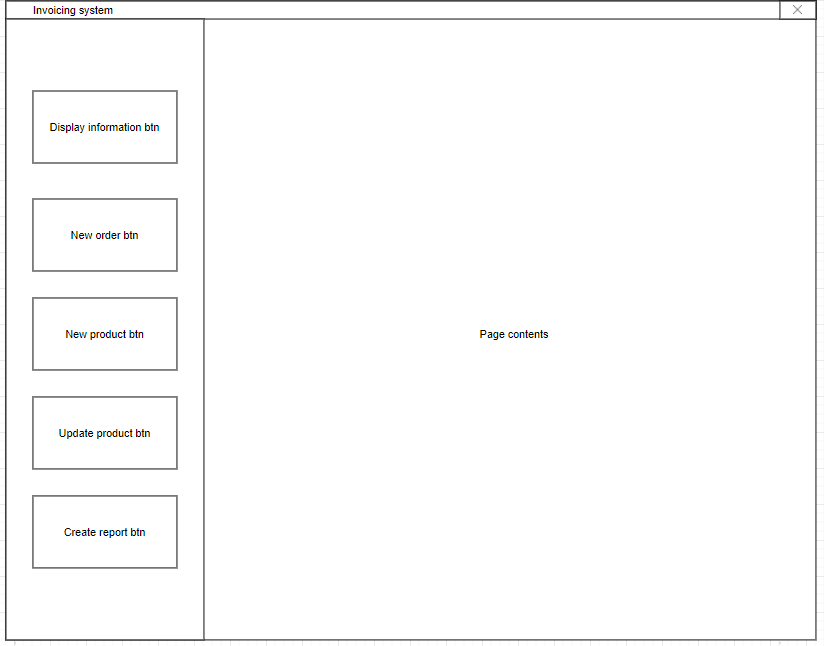


**Login form**

**** 

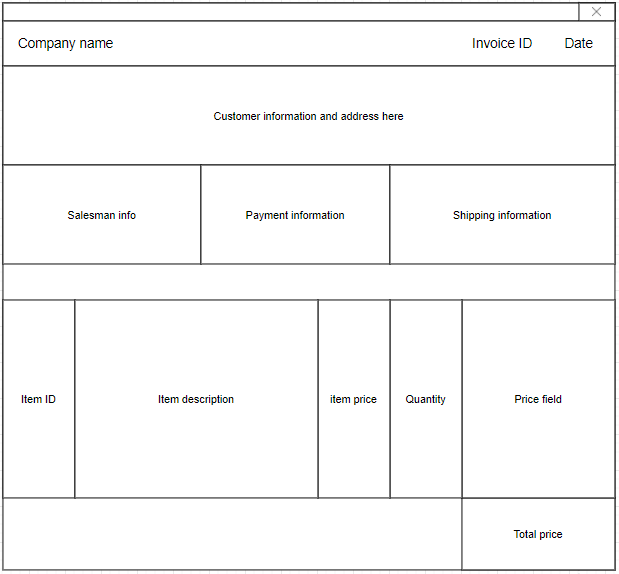
The login UI remained almost identical except the amount of unnecessary white space was minimised. Whilst colours were not incorporated into the wireframes, they were discussed. My client and I agreed upon the colours and I have tried to use them were possible in my design to keep with the theme. The client and I discussed about ideas of how I could potentially do the layout differently and I decided that having less verticality and more horizontal width would be better as monitors are typically landscape orientated. The login panel looks bold and attractive on the monitor.

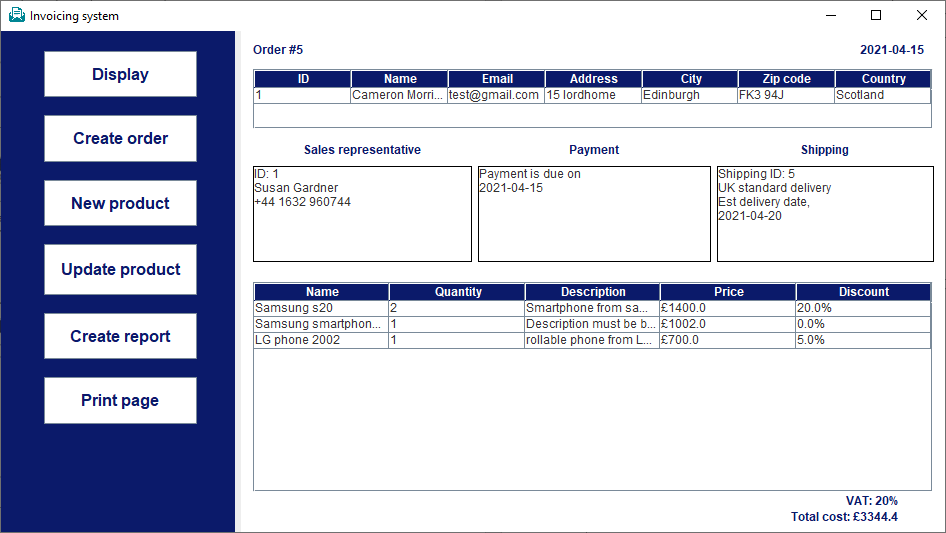
**Navigation bar**



Here we have the navigation design and reality. They are identical except the print button has been added to the navigation bar. I made this decision as having the print button on every single page would take up so much extra room. Having it on the navigation bar also makes it easier for the user to understand what is being printed. The user will now have an understanding that the button on the navigation will print the page contents regardless of what page is being displayed.

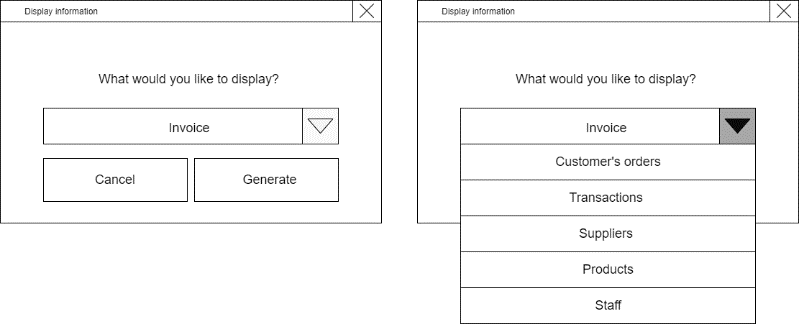
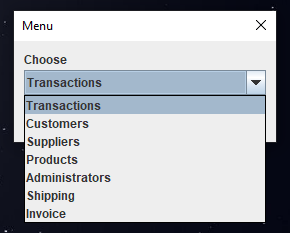
**Invoice page**





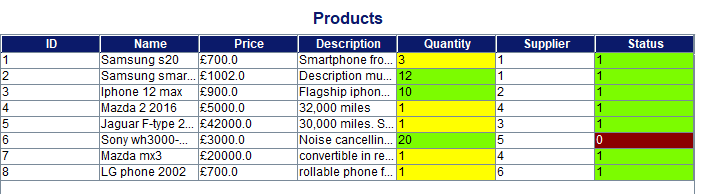
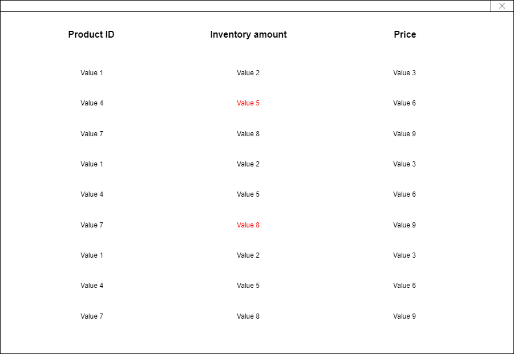
The original and actual invoice pages differ slightly. I removed the company name from the title of the invoice page and instead I just show the invoice ID and date. The customer, salesman, payment, and shipping information are all in the same place as the design. As in the design I left a gap between the details and the order information. The columns of the order table may differ from the design, but the overlying theme remains the same.

**Display selection**



The display section remains identical. I have simplified “Customer’s orders” to transactions. I also changed the name of the staff table to administrators.

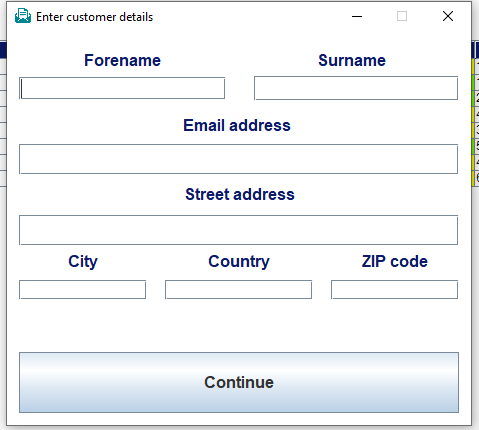
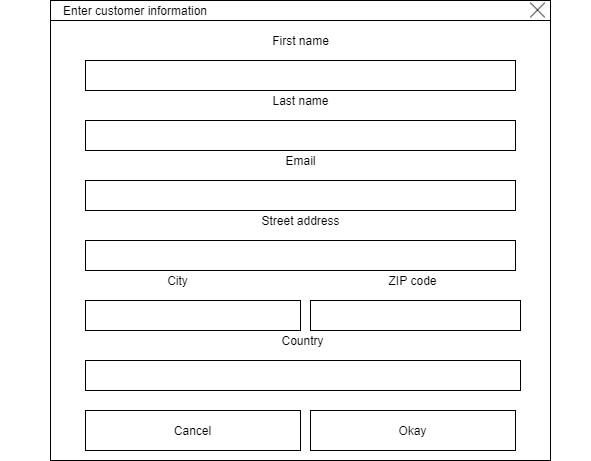
**Tabular view**

****

Here is an actual table, populated with dummy data. This products table shows the actual design I opted for. Unlike the initial design I included a border to each cell and included a title above on the page to tell the user what table is being displayed. In this table, products, the quantity, and status are colour coded. In the design I showed how the coloration could work by changing the foreground colour of one cell to red. However, I noticed upon developing that the contrast of red or yellow on white was simply not enough, especially in small font. So, I instead opted for a different background colour.

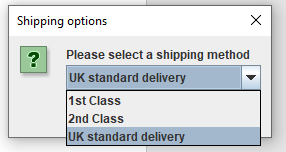
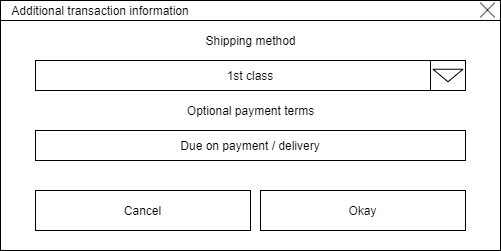
The application includes tabular views for customers, staff members, suppliers, products, low inventory products, orders before date, transactions, shipping information and individual order items. The application focuses on being informative, robust and usable.

**New order form**

****

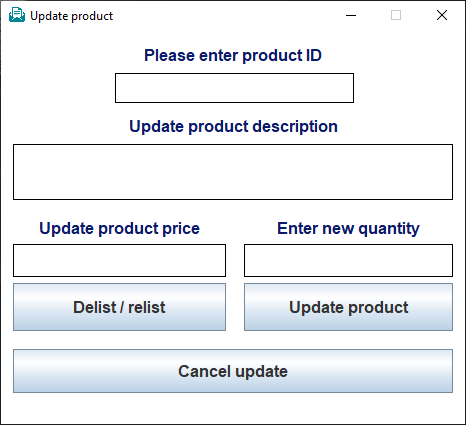
The customer form contains the same information but has been compacted so some smaller input fields share the same line. I have gotten rid of the cancel button as the close button on the top right of the form does the same thing and is where most users typically look to close an application/popup.

**Shipping method form**



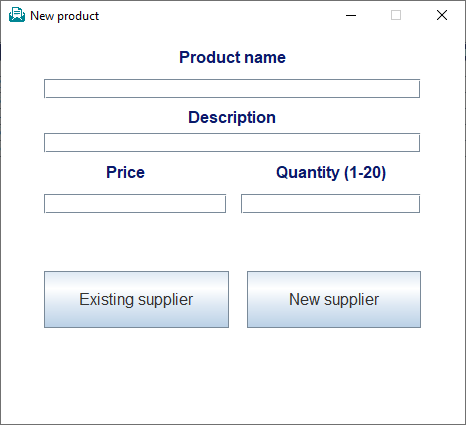
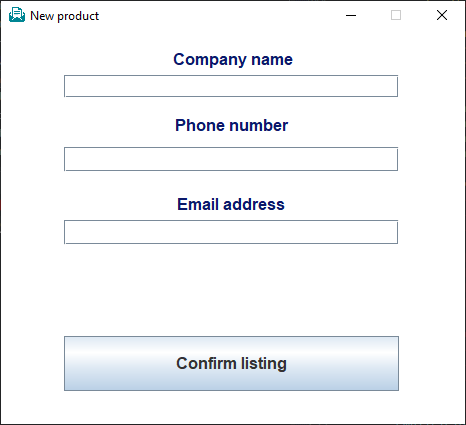
I have removed the optional payment terms as the payment of the order should be soon as possible. I have only included three shipping methods, 1st Class, 2nd Class and standard delivery.

**Update product form**

****

The update form is strikingly like the design except discount percentage has been removed. I have decided to allow the sales representative to decide upon a small discount when convincing the customer over the phone. The discount percentage is now placed in the new order form after customer information is entered. They can set a customer discount percentage if they wish to when making the sale.

**New product form**

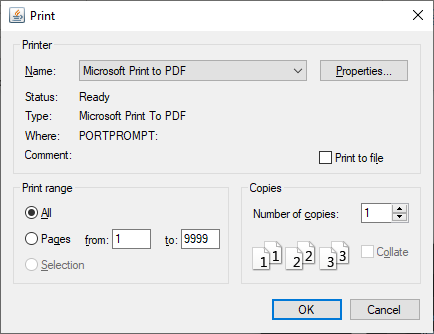
When discussing designs with the client, I mentioned the new product form would have a similar design to the update product form. The design stayed simplistic but unlike the update form the buttons are replaced with existing supplier or new supplier. The picture on the right is the new supplier form.

**Overview page**



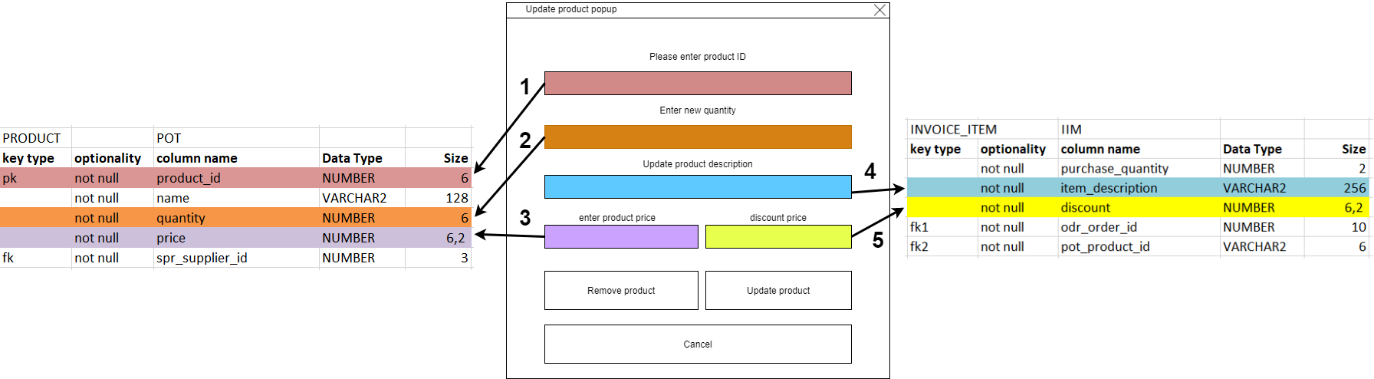
I did not discuss the overview page with the client in detail. The client spoke of wanting to see weekly, monthly, and annual revenue/VAT. Although I did not have a design to reference, I did tick off all the criteria mentioned by the client in our interviews. I also added more information of use such as number of products listed/delisted, employees, products sold, customers, suppliers, and orders.

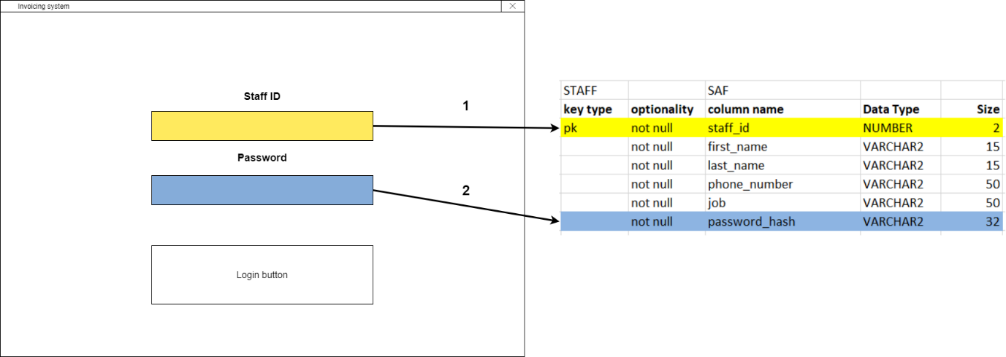
**Printing dialog**



In my design I did not create a sketch for the printing dialog as at the time I thought I would just create a button on the contents page to print the table instantly. I then realised printer settings should be customisable and opted to use standard libraries to create a dialog that would allow the user to print the page contents appropriately.

**Does the design match the business model?**



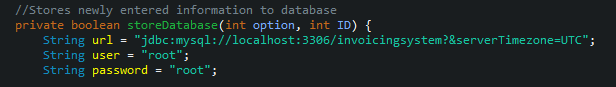
For the update form, the location of description has been moved to the product table. Discount remains in the invoice item table. It occurred to me in the development that having the product description stored in each unique invoice item table was a silly idea and that it should be in the product table which makes it so much easier to update. My thought process in the design was that I would need to reference the invoice item information for the details displayed on the invoice. I got around that problem in Java by using the product ID from invoice item to quickly get the description from the product list.

The design for the login business model stayed the same. The staff ID and the password are stored in the same location. I did not cover any more business models in the design document.

# 3 Use of unfamiliar libraries and/or constructs.

**External libraries**

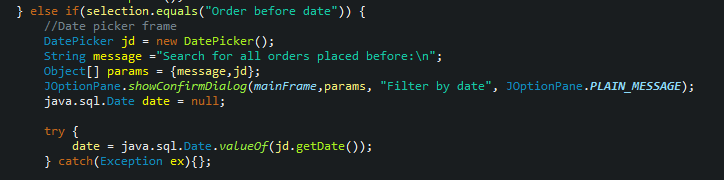
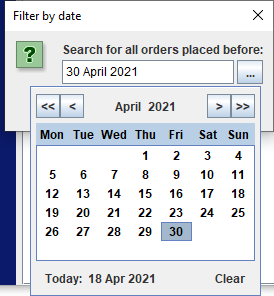
In my program I used two unfamiliar libraries: [MySQL-connector](https://dev.mysql.com/downloads/connector/j/?os=26) and [LGoodDatePicker](https://github.com/LGoodDatePicker/LGoodDatePicker/releases). The MySQL connector allows me to connect my java program to the MySQL command line, so it was essential to this program as it allowed me to store the programs data to a relational database system.

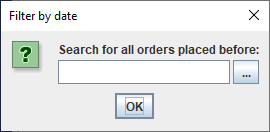




I only deal with the MySQL-connector library in the methods storeDatabase() and loadDatabase(). It allows me to execute MySQL statements using java code to retrieve, store or modify data.

The other library [LGoodDatePicker](https://github.com/LGoodDatePicker/LGoodDatePicker/releases) is a front-end swing component made by GitHub user [WiseEagleOwl](https://github.com/WiseEagleOwl). It allows me to give my client a nice easy to work interface for choosing a date. In my program I use it to select all orders before a certain date.





**Unfamiliar Standard libraries**

In my project I used some libraries that are included as standard with the java development kit that I was not very familiar with. These libraries do not require an install like the previous I have shown.   
I have never worked with any of the [java.awt.print](https://docs.oracle.com/javase/8/docs/api/index.html?java/awt/print/package-summary.html) interfaces. So, I had to do some extensive research in the [java docs](https://docs.oracle.com/javase/8/docs/api/index.html?java/awt/print/package-summary.html) on how to use it. It is a general printing application programming interface (API) that is used to print the contents of swing components.

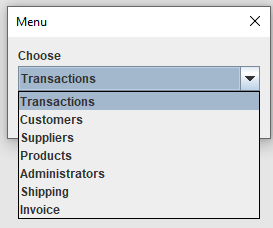
Here are the three interfaces I used to create the printing functionality of my application.



When taking the printing functional requirement into account, I could not imagine what the code would look like and I was relieved that it does not end up being complicated. I simply get use the built-in user interface to allow the user to select their settings; I set the page orientation to landscape as default as my east panel where the data is displayed on my application is more horizontal than vertical. When the user confirms the print button, the custom settings are passed to the print function and the [graphics2D](https://docs.oracle.com/javase/8/docs/api/index.html?java/awt/print/package-summary.html) library gets the measurements of the east panel and start a print job. If something went wrong, then I have added my own try catch statement to display an error message in the event of an exception.



# 4 Error handling

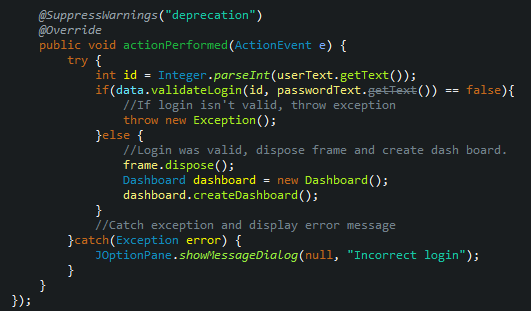
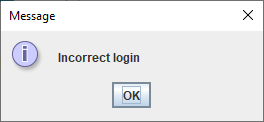
My program contains plenty of validation for all the user inputs. I have used regular expression patterns to verify postcodes, phone numbers, names, and email addresses.   
I have also included error messages and useful user feedback for invalid entries. I utilise warning labels and JOptionPane dialogs to provide as much helpful information to the user as possible.

For the display button the dashboard I have opted for a dropdown selection control.

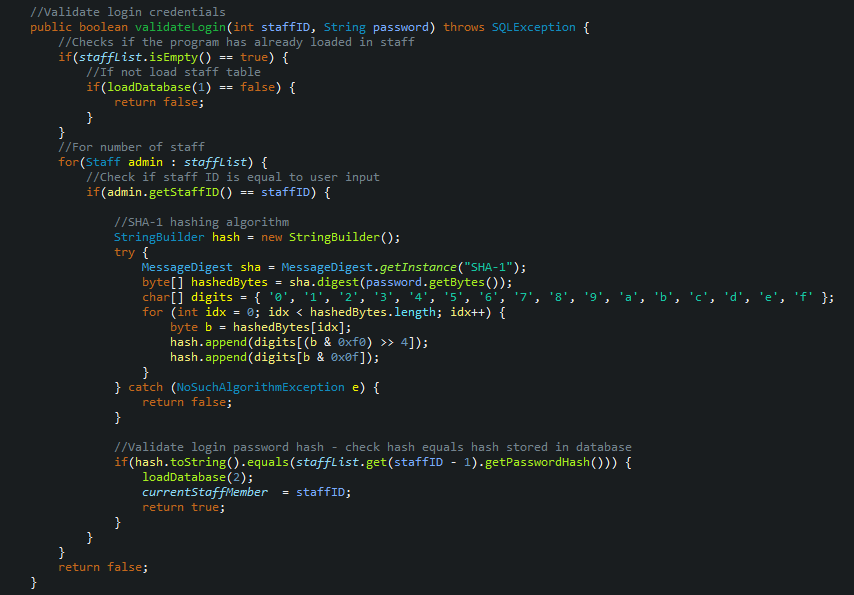
Using a dropdown for this section of the program saves me a lot of time and reduces need for validation.

**Login form validation**

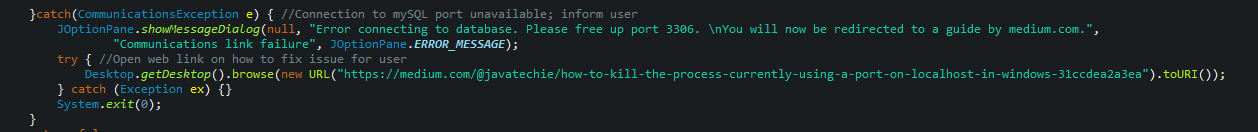
Here is the login form validation and error message. If user enters an incorrect login, throw an exception, and show a dialog.

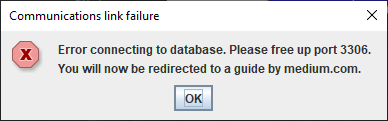


Most of the validation for the login form is handled in the business logic. Once loading in the staff information from the database, it checks if the entered staff ID is valid, if not it returns false and prompts the incorrect login dialog. If valid, it hashes the entered password and sees if the string matches the stored password hash, if it does the login is successful.



When loading in the staff information from the database. If there is an issue with establishing the connection, the program will output an error message and open a browser tab with [information](https://medium.com/@javatechie/how-to-kill-the-process-currently-using-a-port-on-localhost-in-windows-31ccdea2a3ea) on how to free up a port that may be in use.

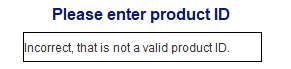
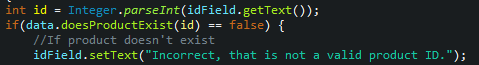




My update form contains quite a lot of validation as you can see. The goal was to create a way of updating 1 or more attributes of an existing product, without being forced to fill in all the empty fields. For example, if you want to update just the price of a product, you can leave the description and quantity fields blank, and they will not be updated.

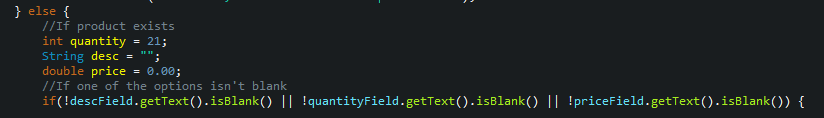
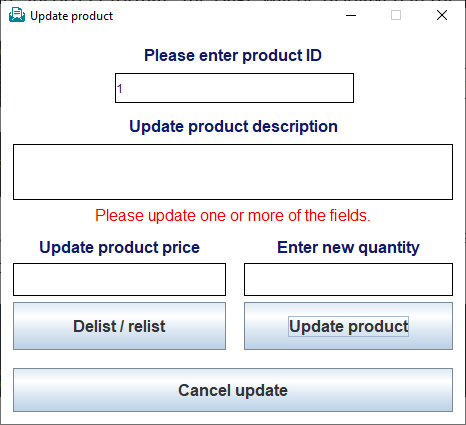


The start of the action listener for update product, checks if the ID exists. If the user enters an invalid ID, the error message is displayed. If they enter a string and the integer variable cannot parse, then an “incorrect format” message will be displayed in the product ID field.



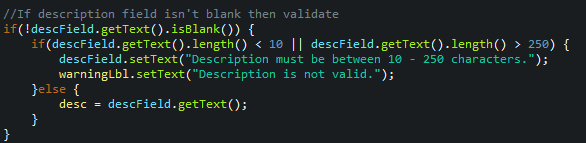


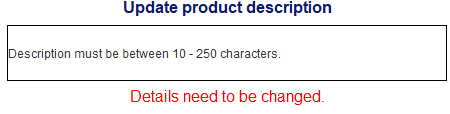
If the product ID is valid, I initialise temporary variables that have invalid values that I will use later. I then check if any of the fields have content entered in them.



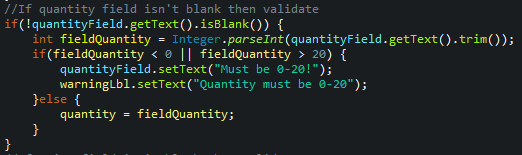
If all three of the updatable fields are empty and the user is trying to press the update button it will then display the warning label with the following message.

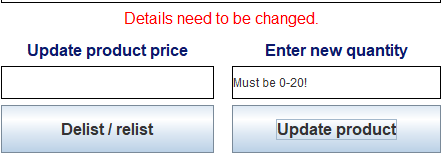
Next, I check if the description is not blank, validate the length of the new content and if the new length is valid, I write over the invalidly initialised description string.



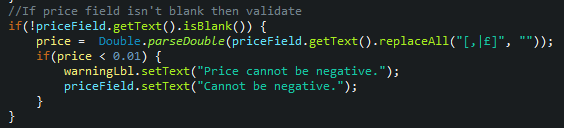
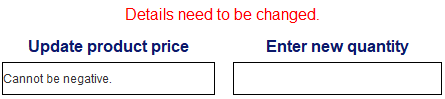
If the description entered is invalid, the UI will display two error messages.

I then check if the quantity field has any contents. If it does, I check if it can parse, meaning if it’s not able to go into an integer variable, I will display an error message about the inputs type seen below. I then check if the quantity is between zero and twenty as the client said 20 was the maximum inventory amount per product.

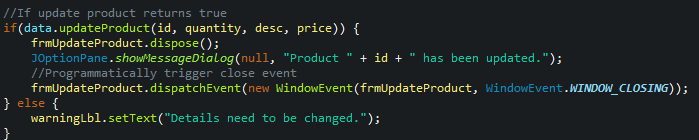


If the quantity is more than 20 or less than 0, the user is told details need to be changed and that the new quantity must be between 0 and 20.

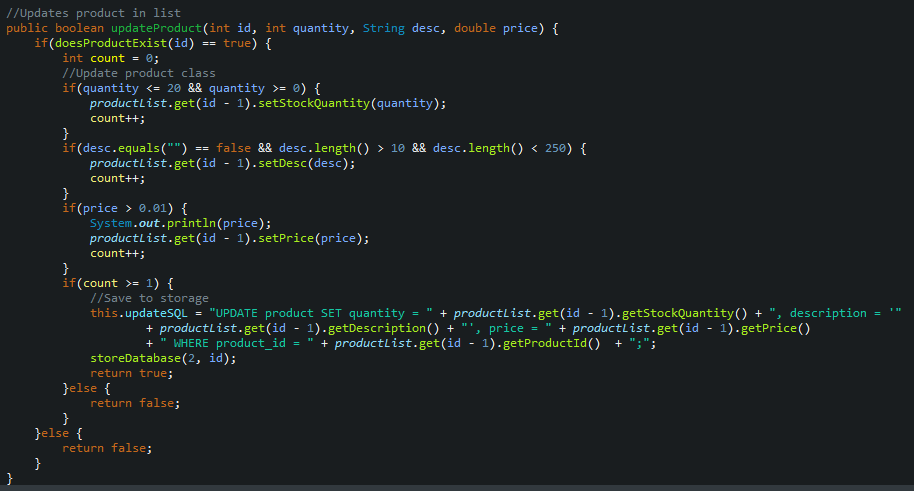
Finally, I check if the price field has user input. I use a regex pattern to remove any potential pound symbol or comma. I then attempt to parse the price, if successful I check if the price is more than 0.01. If not, I display the following error message.



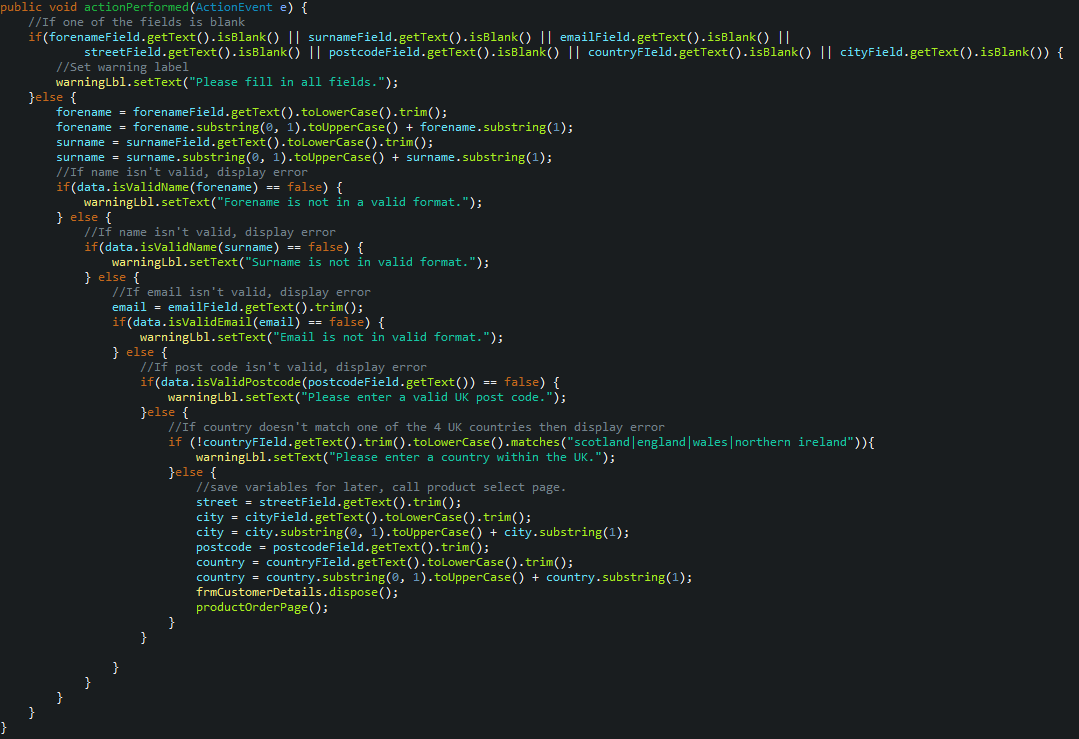
Once the validation for all the fields takes place, I then pass the variables through to the business logic. If the method called returns false, the red label is displayed.

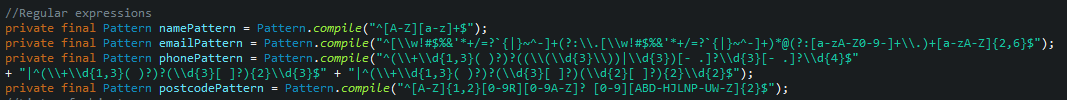


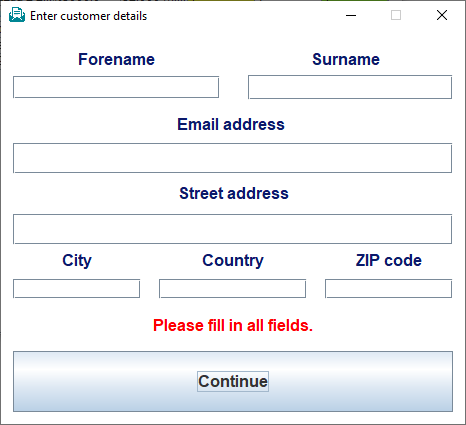
In the business logic, the code is validated to only update new values. Therefore, I initialised the temporary variables earlier with invalid information. I use it to validate what has been updated. I have a count variable which increases if there is a new value. This way the MySQL statement only executes if a value is updated.



**Customer form validation**

****



In the customer form validation, I check to see if one of the fields is blank. If it, is I display an error message. If they are all filled in, I check if the forename and surname comply to name format. I also check if the email, phone number and post comply to the government issued regex pattern. I then convert the entered country to lowercase and if it matches one of the four countries in the UK, I accept it and move on.

If any of the entered information is false, it displays a custom precise error on the UI where the red text on the picture to the right is located message as seem in the code.

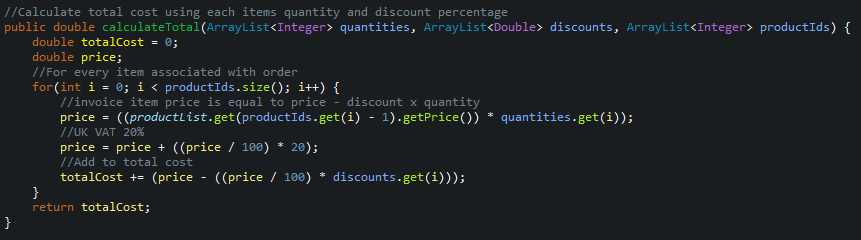
These forms are some of the many examples of where I used validation and selection controls to make my program more robust. My user manual video properly demonstrates the validation of the program and how some fields like price and discount, remove common symbols such as the British pound and percentage sign using regex patterns.

# 5 Internal documentation

Whilst developing my application, I have followed the [oracle Javadoc’s naming conventions](https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html#:~:text=Interface%20names%20should%20be%20capitalized%20like%20class%20names.&text=Methods%20should%20be%20verbs%2C%20in,of%20each%20internal%20word%20capitalized.&text=Except%20for%20variables%2C%20all%20instance,with%20a%20lowercase%20first%20letter.) as closely as possible. My package information is descriptive and indicates who the program belongs to and what the package is for. The packages start with a lower case as standard. My classes all start with a capital letter and are simple and descriptive also. My methods are verbs and camel cased; meaning the first word starts with a lower case and every following word starts with a capital letter.

I commented my code where necessary to explain my thought process and help any future developers work with my code. Not everything was commented as I felt it would be time consuming and cluttered commenting all the well named swing components and their individual attributes.

Here is an example of my internal documentation on this method to calculate the total cost of an order using all the products, quantities, and individual discounts.



I try to keep my commenting simple and used mostly when necessary, to explain calculations or complex code.

|  |  |  |  |
| --- | --- | --- | --- |
| 5. Test plan | | | |
| **SOFTWARE COMPONENT NAME:**  **DATE: 14/04/2021** | | **VERSION NO*:***  **TESTER NAME: Cameron Morrison 569530** | **PAGE NO:** |
| **Test No** | **Purpose/Type of Test** | **Input** | **Expected Result** |
| 1 | Test login details with an invalid staff ID. | Test data 1 | Login failed as it is not an existing staff ID. |
| 2 | Test login staff ID using string instead of numeral staff ID. | Test data 2 | Login fails and as user entered a string instead of a number. |
| 3 | Test login with valid staff ID but incorrect password. | Test data 3 | Login fails as the password entered is incorrect. |
| 4 | Test login with correct details. | Test data 4 | User is taken to dashboard. |
| 5 | Test login with missing no information in the password field. | Test data 5 | Login will fail as an empty string will not match the password. |
| 6 | Test login with no information in staff ID but fill in password field. | Test data 6 | Login will fail as there is no staff ID to search for. |
| 7 | Test new product form with valid information. User enters price as an integer. | Test data 7 | User will move onto entering supplier information if it passes validation which it is expected to. |
| 8 | Test new product with valid information, user enters a pound symbol and comma for price. | Test data 8 | User will move onto entering supplier information as regex statement will remove the pound and comma symbol. |
| 9 | Test new product with a 4-character long description. | Test data 9 | Description is too short to be useful, user will be told to expand upon its length. |
| 10 | Test new product with valid information, user enters a pound symbol and a decimal place. | Test data 10 | Price is stored as a double so the value should be accepted, and the user will move onto the supplier information. |
| 11 | Test new product with Information does not contain a title. | Test data 11 | User will be told to add a title. |
| 12 | User enters “300 pounds” in the price field. | Test data 12 | User will be told that the price must be a number. |
| 13 | User enters valid supplier information into the new supplier form. | Test data 13 | New product will be added, and new supplier will be stored and associated with that product. |
| 14 | User enters an email address without an at symbol into the new supplier form. | Test data 14 | User will be informed email is invalid. |
| 15 | User enters an email address without a domain at the end into the new supplier form. | Test data 15 | User will be informed email is invalid. |
| 16 | Phone number for the new supplier form is gibberish and not a UK number. | Test data 16 | User will be informed phone number is invalid. |
| 17 | The phone number entered the new supplier form contains words. | Test data 17 | User will be informed phone number is invalid. |
| 18 | Information designed for the supplier form is valid except the phone number is does not belong to the UK. | Test data 18 | The program will allow a valid international phone number as sometimes the supplier may be in another country. |
| 19 | User will try to update a product without entering a user ID on the update product form. | Test data 19 | User will be told to enter a product ID. |
| 20 | User will try update the description of a product with a very small description. | Test data 20 | User will be informed the description is too small. |
| 21 | User will try update the price and only the price. | Test data 21 | Only the price for the product will be updated, the other field’s validation will not display warnings. |
| 22 | User will try to set the products quantity above the maximum stock amount. | Test data 22 | User will be told they cannot store more than 20 products. |
| 23 | User will try to update the product with a negative quantity. | Test data 23 | User will be told they cannot store negative products. |
| 24 | User will try set the price of the product as negative. | Test data 24 | Price cannot be equal to or less than zero. |
| 25 | User enters a product ID that does not exist. | Test data 25 | User will be informed that product does not exist. |
| 26 | Test the new customer form with valid information. | Test data 26 | User will progress on to picking products. |
| 27 | Test new customer with invalid name that contains numbers. | Test data 27 | User will be asked to change the name. |
| 28 | Test new customer with a faulty post code. | Test data 28 | REGEX validation will tell that its not a UK post code and inform the user. |
| 29 | Test new customer with a European country (Spain). | Test data 29 | Client specified company only operates within the UK currently. Cannot delivery abroad. User will be informed country is not within the UK. |
| 30 | Test add product form with a non-existent product ID. | Test data 30 | User will be informed that the product ID does not exist. |
| 31 | Test add product form with an unavailable product. | Test data 31 | User will be informed that the product ID does not exist. |
| 32 | Test add product form with 100.01% discount. | Test data 32 | Program will inform user that you cannot give a customer a 100.01% discount. |
| 33 | Test add product form with a quantity higher than current stock. | Test data 33 | Program will tell user there are only so many remaining. |
| 34 | Add product without a discount percentage set. | Test data 34 | Discount percentage will default to 0%. |
| 35 | Test print page button |  | Configuration panel will show and allow user to choose output method. |
| 36 | Test save to PDF functionality. |  | The page you were on before pressing the printer button will be saved as landscape (unless changed otherwise) to a PDF file in the destination of your choice. |

# 6. Test Data

|  |  |  |
| --- | --- | --- |
| **Login form** | **Staff ID (int)** | **Password (String)** |
| **Test Data 1** | 4 | 123 |
| **Test Data 2** | Cameron | 123 |
| **Test Data 3** | 1 | Wrongpassword12 |
| **Test Data 4** | 1 | 123 |
| **Test Data 5** | 1 |  |
| **Test Data 6** |  | 123 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **New product form** | **Product name** | **Description** | **Price** | **Quantity** |
| **Test Data 7** | MacBook air 128GB | Brand new 128GB silver coloured laptop. | 1200 | 3 |
| **Test Data 8** | Samsung tab 6 | Laptop from Samsung 2020 edition in grey. | £1,000 | 2 |
| **Test Data 9** | LG gram | gram | £1,200 | 5 |
| **Test Data 10** | Ipod mini | Classic mp4 device from the early 2000s. | £300.05 | 1 |
| **Test Data 11** |  | This product does not contain a title. | 1000 | 1 |
| **Test Data 12** | Sony headphones wh3000-mx | Noise cancelling headphones 30 hours battery life. | 300 pounds | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **New supplier form** | **Company name** | **Phone number** | **Email address** |
| **Test Data 13** | Sony | +44 2073652810 | help@sony.co.uk |
| **Test Data 14** | Lenovo | +44 2073652810 | Emailaddress.com |
| **Test Data 15** | Volkswagen | +44 2073652810 | vw@hello |
| **Test Data 16** | Tesla | 013201310310 | sales@tesla.com |
| **Test Data 17** | Nvidia | +44 twothreefour | sales@nvidia.com |
| **Test Data 18** | 3M | +202 2073652810 | support@3M-international.com |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Update product** | **Product ID** | **Description** | **Price** | **Quantity** |
| **Test Data 19** |  |  | £20 |  |
| **Test Data 20** | 2 | Phone |  |  |
| **Test Data 21** | 2 |  | £500 |  |
| **Test Data 22** | 2 |  |  | 21 |
| **Test Data 23** | 2 |  |  | -1 |
| **Test Data 24** | 2 |  | -200 |  |
| **Test Data 25** | 15 |  |  | 15 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **New customer** | **Forename** | **Surname** | **Email** | **Address** | **City** | **Country** | **ZIP code** |
| **Test Data 26** | Catriona | Ferguson | cat@outlook.com | 10 sandy lane | Stirling | Scotland | FK2 9PY |
| **Test Data 27** | Camer99 | Morriham | cammer@aol.com | 5 downing street | London | england | FK2 1JP |
| **Test Data 28** | Justin | Dustin | justin@gmail.com | 32 roadingham | Birmingham | England | F22 P23 |
| **Test Data 29** | Ben | Big | big@ben.co.uk | 10 london road | London | Spain | FK8 9PY |

|  |  |  |  |
| --- | --- | --- | --- |
| **Add product form** | **Product ID** | **Quantity** | **Discount (%)** |
| **Test Data 30** | 20 | 3 | 15 |
| **Test Data 31** | 7 | 1 | 20% |
| **Test Data 32** | 1 | 1 | 100.1% |
| **Test Data 33** | 5 | 10 | 20% |
| **Test Data 34** | 8 | 2 |  |

# 7 Test runs

|  |  |  |  |
| --- | --- | --- | --- |
| **SOFTWARE COMPONENT NAME:** | | **VERSION NO: 1**  **TESTER NAME: Cameron Morrison 569530** | **DATE: 14/04/2021** |
| **Test No** | **Actual Result** | **Pass/Fail** | **Comments** |
| 1 |  | Pass | Validation checks if the entered staff id exists and if the entered password matches the associated hash. |
| 2 |  | Pass | There is not a staff ID called “Cameron”. |
| 3 |  | Pass | Password when hashed did not match the hash stored with the valid staff ID. |
| 4 | Information is loaded and the user is redirected to the dashboard. | Pass | When the login is correct, the program loads in the other tables and initializes the dashboard for the user. |
| 5 |  | Pass | Login failed as the empty password string does not translate into the saved hash associated with the staff ID. |
| 6 |  | Pass | The validation is mostly all handled on the business side of my program to avoid client-side manipulation. Due to the staff ID not existing, login is returned as false. |
| 7 | User moves on to supplier information. | Pass | Standard information was entered. |
| 8 | User moves on to supplier information. | Pass | User added a pound symbol and comma which they would be used to do doing elsewhere. Regex removes it and successfully stores the price as double. |
| 9 | User is told to extend the descriptions length. | Pass | Less than 10 characters in length is extremely short and unlikely to be useful as a description. |
| 10 | User moves on to supplier information. | Pass | Program accepts and parses pound symbols, commas, or decimal places. |
| 11 | User is told “Please fill in all fields”. | Pass | If one of the fields is empty the validation will notice that and display a red warning label. |
| 12 |  | Pass | “300 pounds” is not a way of formatting price, the program accepts a number or great British pound formatting. |
| 13 |  | Pass | Phone number and email comply to regex patterns. |
| 14 |  | Pass | Email is missing an @ symbol. |
| 15 |  | Pass | Email does not contain a domain so is declined. |
| 16 |  | Pass | Phone number was made up and did not comply to the mobile number format. |
| 17 |  | Pass | Phone number contained characters. |
| 18 |  | Pass | International phone numbers are accepted. |
| 19 |  | Pass | To update a product there must be a product ID. |
| 20 |  | Fail | Product does not actually update due to the validation in the business logic but for some reason instead of showing an error message, it says the update was successful. |
| 21 |  | Pass | Pound symbol is removed before parsing double data type. |
| 22 |  | Pass | User is informed of the acceptable range. Number cannot be more than maximum. |
| 23 |  | Pass | User is informed of the acceptable range. Number cannot be negative. |
| 24 |  | Pass | Price of an item cannot be updated to a negative value. |
| 25 |  | Pass | Validation checks to see if the list contains the number and returns false if it does not exist. |
| 26 |  | Pass | The correct information successfully passed through the validation. |
| 27 |  | Pass | Forename cannot contain numerals. |
| 28 |  | Pass | Test data postcode does not comply to government standards. |
| 29 |  | Pass | Spain is not one of the four options unfortunately. |
| 30 |  | Pass |  |
| 31 |  | Pass | If status is 0 then it is delisted currently. |
| 32 |  | Pass | Discounts cannot be negative or more than 100% |
| 33 |  | Pass | User tried to order 10 but instead got informed of the number in stock currently. Which is only 1. |
| 34 | Product successfully added to basket. Moves onto shipping information. | Pass | Discount percentage defaults to zero percent when left empty. |
| 35 | Configuration panel for printer is displayed. | Pass | Contents of what is currently being displayed on the program excluding navbar can be printed/saved. |
| 36 | Windows 10 save to destination dialog pops up. Once file is named and stored, PDF opens with the contents of the page saved landscape on it. | Pass | PDF can then be sent to others or printed. |

1 Failed test (20) now passes when retried. I have noted the issue and fix in my logbook and work diary.

# 8. User documentation

I have created a user manual recording with audio. The video is a walkthrough of how to work the program. It also demonstrates all the program’s input, outputs, and validation.

The video is named “User manual” and will be placed alongside this document in the provided folder.

To set up the application, the user must download MySQL shell from the link below.

<https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-shell-install-windows-quick.html>



Figure Taken from the oracle install guide.

Open the file, accept the terms and conditions then choose custom install.

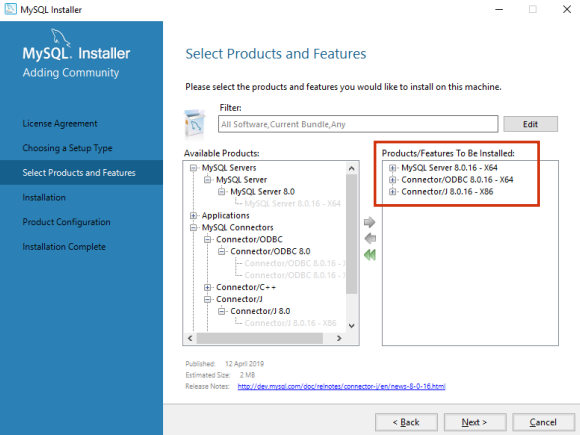
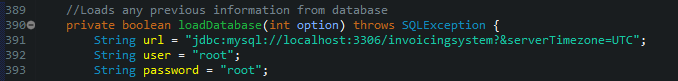
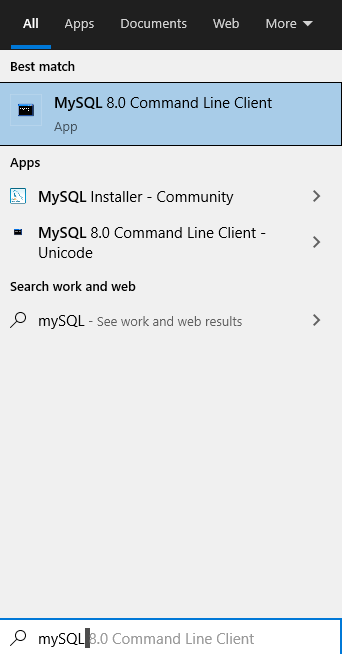
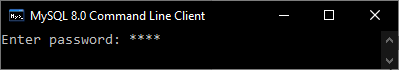


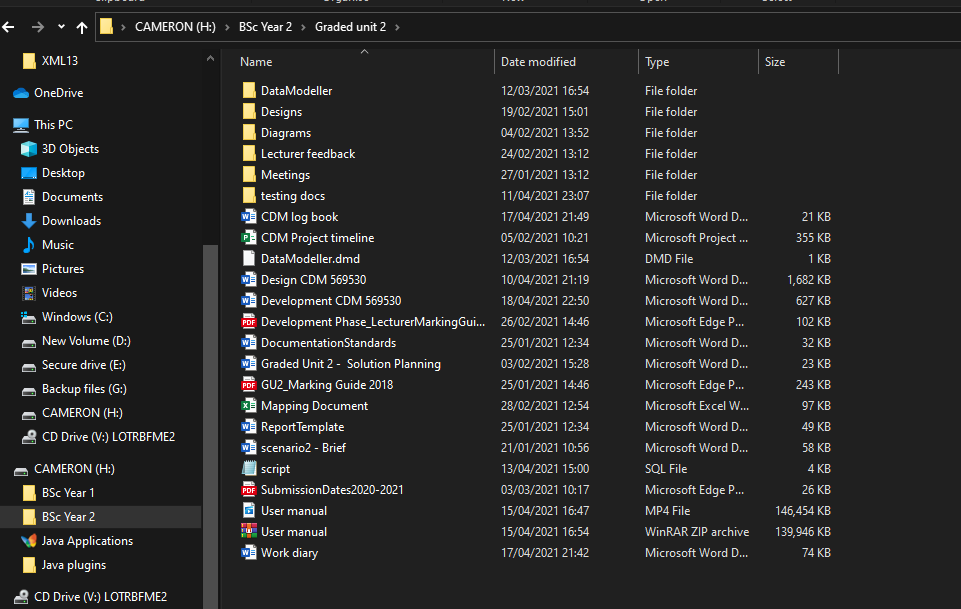
Figure Taken from the oracle install guide.

From the available products, select the three highlighted on the right and click next. When prompted to create a database password, set it to the default “root” or for added security, create your own. Doing so however would need you to modify the password variable in the DataManager methods: loadDatabase() and saveDatabase()

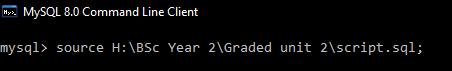


Once the file has downloaded, search “MySQL” into your operating systems search bar and open the application. If you did not set a password the default password will be “root”.



Once you are logged in, open file explorer and head to the download location of the install files. For you it will most likely be the download folder. At the top of your install location, click on the folder icon and it will display text. Copy the text.

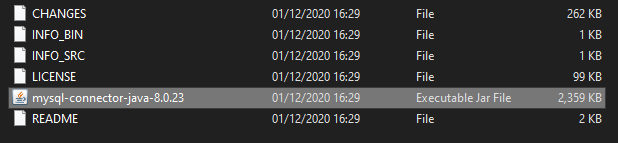
Now return to the MySQL command line client and type “source”, then paste in the copied text and add to the end “\script.sql;”. Please see my example below. Press enter.

 You should now be good to open the jar or executable file.

The two unfamiliar libraries used in the design have been packaged in the external jar file. You should not need to install them again but for development purposes I will demonstrate just in case another developer decides to improve upon my work.

For the MySQL database connector once the community edition database is installed. Its folder should contain a jar file.

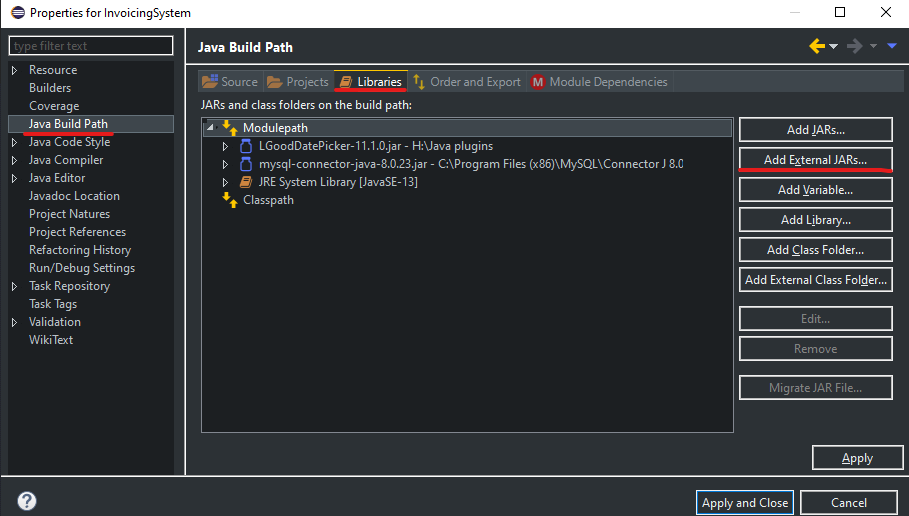




To download LGoodDatePicker, go to the [github source here](https://github.com/LGoodDatePicker/LGoodDatePicker/releases/tag/v11.1.0-Standard). Download the jar file seen in the image below. Remember the location of the download for later.



You now simply go to your code integrated development environment; I am using Eclipse. Right click on the top-level folder of the project in your editor and go to properties. Locate where your editors build path is. Click on Libraries. Then click on add external jars. This process will apply to both MySQL connector and LGoodDatePicker. When you have located your jar files and added then, finally click apply and close.



# 9. Bibliography

All images in this document except the figure images are my own intellectual property.

1. Java doc naming convention

<https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html#:~:text=Interface%20names%20should%20be%20capitalized%20like%20class%20names.&text=Methods%20should%20be%20verbs%2C%20in,of%20each%20internal%20word%20capitalized.&text=Except%20for%20variables%2C%20all%20instance,with%20a%20lowercase%20first%20letter>.

2. MySQL command line

<Https://dev.mysql.com/doc/mysql-shell/8.0/en/mysql-shell-install-windows-quick.html>

3. Unfamiliar libraries: MySQL connector, LGoodDatePicker.

<https://dev.mysql.com/downloads/connector/j/?os=26>

<https://github.com/LGoodDatePicker/LGoodDatePicker/releases>

4. Unfamiliar standard libraries: java.awt.print

<https://docs.oracle.com/javase/8/docs/api/index.html?java/awt/print/package-summary.html>

<https://docs.oracle.com/javase/8/docs/api/index.html?java/awt/print/package-summary.html>

5. MySQL connection issue website.

<https://medium.com/@javatechie/how-to-kill-the-process-currently-using-a-port-on-localhost-in-windows-31ccdea2a3ea>

6. Figure images taken from Oracle academy MySQL install guide.