Student Name: Lee Taylor Student Number: 190211479

Nucode Repository: https://nucode.ncl.ac.uk/scomp/stage1/csc1035/teams/Team-53/assessment-3-team-project---small-business-case-study.git

An overview of how the team was organised and managed

During a practical, the team was organised according to the distribution of tasks decided by a team discussion, in-person we coordinated which jobs to undertake based on personal preference and skill whilst being conscious of others. We utilised Microsoft Teams to communicate and further manage our team by producing, updating and maintaining a task-document stating for each task a description, current status, and person assigned – this enabled us to effectively communicate each person's task and responsibility coherently, as a change made to this document would be seen by all members enabling us to further communicate any changes in our current task, and if anyone started a new task. (Below is a snippet from the document)

- (Optional) Allow user to input a stock item into the database ✔ [Lee]
  + User may input a stock item into the database
- (Optional) Output an ASCII table displaying all the stock items in the database ✔ [Sofia]
- (Optional) Output contents from the database to a CSV file ✔ [Tiger]
- A way of producing a receipt of the transaction in the EPOS system ✘ [Douglas]

Figure 1 - Snippet from task-document

In terms of our project-development workflow, we chose to create and maintain our own individual branches, this was because we wanted to avoid creating clashes or conflicts and to not waste development time by asking and waiting for permission to upload any work. I simply agreed to merge and implement everyone's work from their personal branches near the end of development – this process was made simple as I kept track of everyone's work through our task-document allowing me to plan-ahead.

A brief summary of your contribution(s) to the case study and some of your coding solution(s)

The first methods I developed allowed the modification of stock values outside a transaction, allowing increasing, decreasing and the setting of the stock-value of an item. Developing these 3 methods allowed me to gain a deeper understanding of how to modify the database using the libraries javax.persistence.Query and Hibernate-native-SQL to create a query, then within the newly created query, create and set parameters to the user input and then finally execute the query. I then shared this code with my team via Microsoft Teams in order for them to also communicate with the database. Based on the first methods I developed I was then able to develop more features such as allowing the creation and insertion of stock-items into the database and the deletion of stock items from the database.

Later in development, based on the client's specification I concluded that we may need a separate table for storing each transaction – based on this I drafted an ER-diagram (Figure 2)

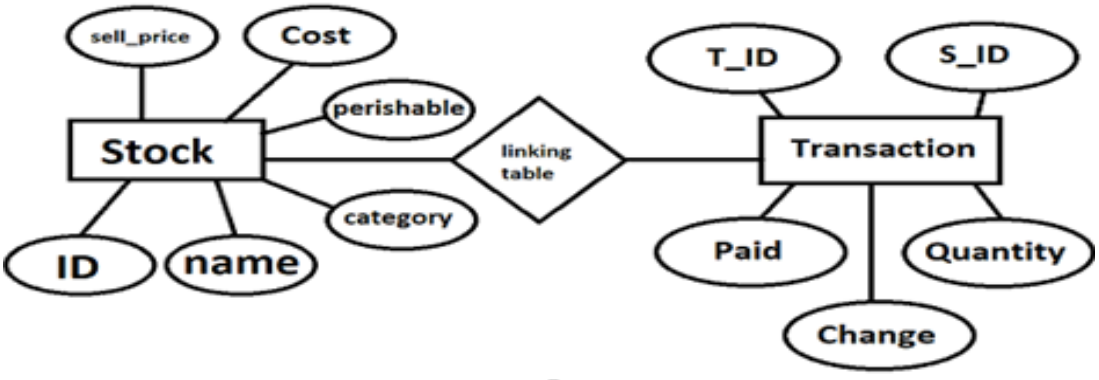

Figure 2 - First draft of ER diagram

I discussed the proposition of another table with my teammates, to which they agreed is necessary, and then presented them this diagram for peer review. We decided the above ER diagram is wrong, so I then developed another solution, this time denoting the actual tables themselves whilst factoring in to remove atomicity, multi-valued attributes, and, partial and transitive dependencies, as producing a database adhering to third normal form is vital to reduce redundant data, prevent data anomalies and to ensure referential integrity. I then checked with my teammates to which we agreed to the new design. (Figure 3)

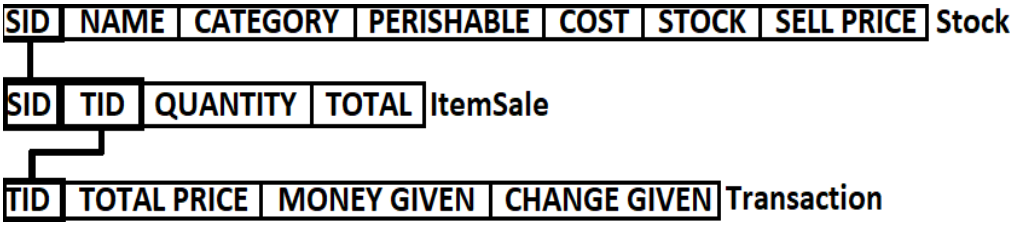

Figure 3 - Second draft of Database design

I then went onto develop, the UI to allow the user to communicate with the program, some commands for a customer transaction, methods to extract information from the database for validation and output, a "CRUD" interface to significantly reduce code duplication, the ability to modify the stock of an item outside a transaction, the ability to see a count of the available stock and merged everyone's classes and methods near the end of development. Whilst helping with Sofia's CSV reader to instead of inserting invalid data into the database when reading an incorrect data entry, it simply goes to the next entry and then modifying it to not insert duplicate entries, informing Tiger Kato on methods of how to write to a file and how to read data from the database, and lastly implementing necessary functionality within Douglas' receipts, to also present the items bought in a transaction.

Reflection on your contributions to the team and how it will affect your future teamwork

The following reflection adheres to (Gibbs' reflective cycle, 2020).

Describing and *reflecting* on my overall responsibilities and contributions to the team, I began with the task of "modifying stock outside a transaction" I abstracted and decomposed my task in order to simplify my objective. Which required me to learn how to use libraries and methods I was unfamiliar with, which I found quite intuitive, as focused research led me to find multiple providing satisfactory explanations. This enabled me to develop a solution relatively quickly and fulfil my responsibility at the time. However based on my previous teamwork in semester 1 I felt that I did not help my teammates enough back then, so I then took on the responsibility of developing extra features, developing more specification requirements and merging everyone's work at the end, whilst attempting to work with my teammates on their methods. During development I *felt* very pleased that most of the specification requirements were met early on, I also felt that I could communicate very well with my teammates as we were able to coherently describe exactly what we were working on and what we needed help with, which enabled me to offer very specific advice. However, I did experience high amounts of stress near the end of development, after my teammates had completed their part of the specification, having to implement everyone's work together occurred quite close to the deadline which invoked a lot of pressure, which could have negatively affected the final result of our program. However, *evaluating* the current state of our project displays that the requirements were met with some extra features, which in my opinion demonstrates our communication was effective and efficient as we were able to carry out and fulfil all of our responsibilities as a team. I felt that my repeated use of thinking abstractly and decomposing my problems throughout the project in conjunction with extensive research allowed me to develop solutions relatively quickly and efficiently, enabling me to spend more time working with my teammates. However, the high amounts of stress and pressure I experienced during merging the project could have negatively affect the quality-of-service for my team's product, which could have been avoided if I had delegated some of the merging or if we had worked on a single branch. Therefore my *action plan* for when working with another team in the future, is to continue carrying out focused and extensive research into topics relevant to the project at-hand whilst continuing to decompose and abstract my problems in order to simplify them and develop solutions more rapidly in order to fulfil my responsibilities. Whilst also attempting to maintain the same, if not better, level of communication and coordination with my future team. Lastly, I will also investigate the workflow of developing on a single branch in a team as this workflow could prevent the high-pressure, I experienced near the end of development in order to minimise the quality-of-service from being negatively affected. In conclusion however I believe I could improve my teamwork by considering to use a different type of git-workflow. I felt I worked well with my team in terms of communication and coordination, as well as working on my own to fulfil my own responsibilities to the team. (What is abstraction? - Abstraction - KS3 Computer Science Revision - BBC Bitesize, 2020) (What is decomposition? - Decomposition - KS3 Computer Science Revision - BBC Bitesize, 2020)

Personal Log of Work Undertaken

File/Method Created or Modified	Date	Description of work
Created `updateStock()` method in Main.java	26 / Feb / 2020	Created and designed a method that creates an SQL query to changes the stock value of an item in the database.
Created `increaseStock()` method in Main.java	26 / Feb / 2020	Created and designed a method that creates an SQL query to increase the stock value of an item in the database. By extracting the current stock value of an item and then performing addition and then updating the stock value of the item in the database.
Created `decreaseStock()` method in Main.java	26 / Feb / 2020	Created and designed a method that creates an SQL query to decrease the stock value of an item in the database. By extracting the current stock value of an item and then performing subtraction and then updating the stock value of the item in the database only if it is not negative.
Created `Stock` Class.	26 / Feb / 2020	Created a class that links to the table "STOCK" within the database.
Created database table design.	27 / Feb / 2020	We wanted to keep a record of all previous transactions that have taken place, in order to store such information, we decided to use more than one table. I drafted a design stating which tables should be created and what the tables should store which was then checked and approved by two other team members and implemented by another team member.
Created `insertStock()` method in Main.java	28 / Feb / 2020	This method takes 6 String inputs from the user which are the 6 attributes

		of a stock item. This method as of yet does not insert the stock-item into the database, the validation of input still needs to be perfected.
Created Methods.java class.	28 / Feb / 2020	This class is used to house all methods currently being developed by myself.
Updated method `insertStock()` method in Methods.java	28 / Feb / 2020	This method now properly validates all user inputs and disregards the stock item depending on the degree at which the user input is invalid. Furthermore, I still need to implement inserting a valid stock item into the database.
Created method `extractItemNames()` method in Methods.java	28 / Feb / 2020	This method extracts all item names from the database and stores them in a hash map. Where both the key and value is the item name.
Updated Main class.	28 / Feb / 2020	Updated the main class to allow the user to choose options on how to interact with the system. Acting as a beta-version of the EPOS system to build.
Updated Stock.java class.	29 / Feb / 2020	Changed the tags and syntax according to Jordan Barnes' announcement in lecture. Pertaining to compatibility issues.
Added new class customerTransaction.java.	29 / Feb / 2020	Added placeholder-structure for customer transaction optional-inputs. No functionality exists as of yet.
Reorganised project structure.	29 / Feb / 2020	All methods held in classes that are still being modified, developed and tested have been moved within a newly created package/directory named testing – to follow the specification. Most methods have been moved from the methods-class and into their own corresponding classes.
Updated method `extractItemNames()` method in Methods.java	29 / Feb / 2020	The HashMap used to store all item names, now

		stores all item names alongside their respective stock values as keys. (I later realize this can be used as a count for available stock)
Updated (beta-EPOS) Main class.	29 / Feb / 2020	User interaction with the system has been extended to allow the user to modify stock values of items in the system from the Main class.
Added `buyItem()` method in CustomerTransaction class	1 / Mar / 2020	Added functionality for a user buying an item such that the item existence is validated and the amount to purchase does not exceed the amount of stock stored in the database. A user can run this method as many times as necessary to add all items to a transaction.
Added `viewCurrentShoppingList()` method in CustomerTransaction class.	1 / Mar / 2020	Added customer transaction functionality as a user can now see how many items are currently in transaction, the quantity and the price of each item based on the quantity to purchase. No total price can be seen which must implemented. This current list of items in transaction is printed out as an ASCII table.
Added `removeItem()` method in CustomerTransaction class	1 / Mar / 2020	A user may remove an item from a transaction currently open. Such that if a user adds items x,y and z to a transaction then a user may specify any item from the current list to remove.
Updated `beginCustomerTransaction()` method in CustomerTransaction class	1 / Mar / 2020	Updated this method to account for the new methods created above. Also added the functionality in this method to cancel an open transaction.
Implemented `viewCountOfAvailableStock()` method within newly created class Output.	2 / Mar / 2020	Added a method to display the contents from a previously made HashMap, item names and their stock, in a pretty

		format that being an ASCII table.
Implemented CRUD interface and class “CreateReadUpdateDelete”	3 / Mar / 2020	Added and re-wrote the method definitions for the CRUD interface provided from using objects, to using HQL instead. Also re-wrote/updated existing methods to use CRUD interface-methods, code duplication was significantly reduced.
Added feature “search item”	5 / Mar / 2020	Whilst a customer transaction is open, the user may search an item by name and its price and stock are returned.
Implemented Other Team member’s method.	5 / Mar / 2020	Tiger Kato wrote a method, I downloaded and made this method accessible from the main menu of the program.
Implemented Other Team member’s method.	5 / Mar / 2020	Sofia Trevino wrote two methods. I downloaded and modified these methods to be accessible from the main menu of the program.
Wrote and added (WIK) LICENSE and README files.	6 / Mar / 2020	Wrote README to explain most of the program, and stated authors, still need everyone’s student number and write a testing section.
Uploaded my repository to the master branch	10 / Mar / 2020	I downloaded the origin/master branch and then merged this branch with work from my branch
Modified master branch	10 / Mar / 2020	I moved all of my classes from the “Testing” directory to the main directory
Merged master branch	10 / Mar / 2020	Performed further merging with Douglas’ and Josh’s branches and fixed some small bugs I found immediately
Added functionality to receipts	11 / Mar / 2020	The method to write receipts, designed by my teammate did not display each item, quantity and price on the receipt, I noticed this and added this functionality to the receipts

Added alternative generation of IDs	11 / Mar / 2020	The current method to generate transaction IDs used generating random numbers and comparing to the list of current IDs, in theory this would become extremely inefficient at some point due to the method of generation. I wrote an alternative that generated IDs more efficiently and would never become inefficient.
Added method `payForTransaction()` in csc1035.project3 in class CustomerTransaction	11 / Mar / 2020	This method re-writes the structure of customer transactions by making the customer pay for transactions before any database-transactions take place, therefore if the customer did not have enough money to pay for the items they could now go back and remove an item.
Restructured classes and methods	11 / Mar / 2020	Reduced the number of classes by moving methods from classes into already existing classes
Added Sofia's new method `toTXT(…)`	11 / Mar / 2020	Sofia wrote a method to write the contents of the database to a text file, with some of her methods already in the program I simply copy and pasted her new method to avoid merging conflicts and complexities
Added method `outputMenu()` in class Main	11 / Mar / 2020	The main menu seemed crowded with too many options, the number of options was reduced by redirecting some of the options behind another menu

References / Appendix

BBC Bitesize. 2020. *What Is Abstraction? - Abstraction - KS3 Computer Science Revision - BBC Bitesize*. [online] Available at: <https://www.bbc.co.uk/bitesize/guides/zttrcdm/revision/1> [Accessed 13 March 2020].

BBC Bitesize. 2020. *What Is Decomposition? - Decomposition - KS3 Computer Science Revision - BBC Bitesize*. [online] Available at: <https://www.bbc.co.uk/bitesize/guides/zqqfyrd/revision/1> [Accessed 13 March 2020].

The University of Edinburgh. 2020. *Gibbs' Reflective Cycle*. [online] Available at: <https://www.ed.ac.uk/reflection/reflectors-toolkit/reflecting-on-experience/gibbs-reflective-cycle> [Accessed 13 March 2020].