**What types of relationships are currently written into the design?**

In our grocery store system we currently have the relationship of inheritance, composition, and aggregation.

**What types of relationships are not written into the design? Why?**

We do not have any composition relationships written into the design because all of our classes are able to function by themselves and are not reliant on other classes. We chose to not add compositions because we wanted all the classes to be not reliant on other classes so that our program would be less fragile and less prone to unexpected errors caused by editing different classes. We also did not see a need for a composition relationship in our program.

**What was an example of an alternative design you considered? Why was this way chosen?**

An alternative design we considered using a class called “grocery store”, to be the super class for all the other classes, instead of our current design in which the core classes are not inherited from each other, but rather connected through aggregation and association. We initially planned on having a more rigid class structure where all the classes will inherit attributes from the grocery class, in order for our program to emulate that of a physical grocery store. However we later decided to

we instead planned on having the core classes based off a physics

**Which classes will be created? Which ones will not? Why were these ones chosen?**

The classes we created during the first sprint are the account class, cart class, profit class, and item class. The reason we chose these classes is because there are the core classes of the program in which the other classes will depend on. The account class was created during the first sprint because all the other classes will use information from the account class as all the information inputted from the user will be processed and stored in the account class. The cart class was created first because all the purchases will be made and stored through the account, which is obviously an essential function for a program based on an online grocery store.The profit class was created during the first sprint because being able to calculate profit is essential for the cart class and item class to function. The item class was created during the first sprint because the item class is the super class to three other classes that will be created in the later sprints.

**What values and variables will be created for this “first sprint”, of only 4 days? Why are these prioritized, and others omitted?**

Obviously with the first sprint of our system, we would like to have values and variables that are crucial to the functionality of a grocery store. The classes that we believe are the most important to the system include the account class, cart class, item class and profit class.

(account, cart, item, profit)

We created all the values and variables within the listed classes that were completed during the first sprint because the variables within these classes were dependent on each other and were essential for the said classes to function as intended. However, we may add new variables or attributes when we create the other classes in the next sprints so that already created classes will work with the classes we plan to create in the next sprints.

**How would you plan out the different sprints? What would be a reasonable goal for the next two sprints based on the current design?**

We planned out the different sprints by first clearly organizing the different objectives and classes to complete within the sprints and what time frame should the sprints be completed. We also planned out who will be doing what for each sprint, and most importantly, how we will make sure the code written by one group member will work with the code written by another group member. Once we neared the end of the current sprint, we decided how the current sprint and the next sprint will come together and work towards improving the current program. We also made sure that the objectives we completed during the first sprint could be modified without hampering the overall program as it is essential for the code to leave room for modification and new associations that would inevitably happen during the later sprints. A reasonable goal for the next two sprints based on the current design is to make sure that the different associations between the classes function as intended after the next sprints. We will also make sure that the different classes are able to work together in the program and that the overall program will have a clear representation of the class diagram we first made for the program. Each program will have the same functionality and attributes as we first planned out so that the different classes will work as intended.

**What achievables do you expect for these sprints?**

From these sprints we expect to receive valuable feedback on our code in terms of usability. We will also understand what errors we may possibly encounter with our code. We plan to achieve a system without any errors at the end of our sprints. We also plan to have a working program which will demonstrate the different class structures and associations as well outlined in the UML diagram. We will also make sure that we achieve a proper coding style for our code such as adding a toString to each class, making sure the super class is able to use the different sub classes, and making sure that the different classes could be made into objects for use in other classes.

**What dependencies exist in your system? How did that help you prioritize which ones to create?**

As we outlined in our UML diagram, many dependencies exist in our system. In our system, the customer service class and the cart class are dependent on the account class to function. The cart class is dependent on the account class because in order for a cart to exist, the customer needs to create an account to store the cart within to buy stuff. The cart is connected to the account class through an attribute in the account class which acts as an object of the cart class, so that the account class will always be able to access the attributes and functions of the cart class so that each account can always create a different cart for its own needs. The customer service class is dependent on the account class because in order for a complaint or request to be made, an account needs to exist in order to facilitate it. The account class is connected with the customer service class through an object in the account class that is not inside of its constructor which allows the account class to use the customer service class. The dependence of the cart and customer service class on the account class led to our decision to prorize the completion of the account class before the cart and customer service class is completed. We also based the functionality of the cart and customer service class to be compatible and fit the requirements in order for it to work with the account class. Another dependency in our system between the item class and the cart class. The cart class could add items into the cart and use the attributes in the item class. The cart does this by using an array with the data type as item, so that the cart could hold different items inside the array, and each element inside the array could use the attributes of the item class. This dependency helped me decide to first create the cart class and plan the item class to make sure it works with the cart class. The final major dependency in our uml system is the inheritance relationship between the food, clothes, and electronics class with the items class. The items class acts as a super class for the clothes, food and electronics class. This dependency helped us decide to first create the items class and then create the clothes, food and electronics class in later sprints and to make sure the said subclasses worked with the item super class. Through all these dependencies, we have decided that we will first create the account, cart, and item classes during the first sprint as they are the most essential classes in which other less important classes depend on to function.

**Who will be in charge of creating each class?**

We tried to split the programming and creation of each class equally. Matthew will be creating the Account class, Profit class, Electronic class, as well as testing the main class a lot. Ryan will be creating the Clothing class, Item class, and Employee class. Jordan will be creating the class CustomerService, Cart class, and Food class. We decided to split the classes up like this so there is an even distribution of workload amongst the members of our group.

**How will you ensure that in the end, the program will work?**

Getting a program to work with the combination of multiple group members and multiple classes is definitely not an easy task. We tried to combat such issues by planning out the UML well and easy to understand. This allows for the group members to easily follow what should be done and how each variable name should be spelt so it doesn’t crash with the classes of other members. Obviously even with these precautions, it is almost guaranteed that our classes will have errors working with each other. This is where we will be constantly testing and fixing errors of our code. We can’t guarantee that our program will work together in the end but we can reduce the chance and amount of errors we will have when combining our classes. This is how we’ll ensure that in the end, our program will work together.

**How will you demonstrate that the classes are working? How will you test that each part is complete?**

We demonstrated that our classes are working by running and testing through the program and intentionally trying to look for errors that can occur from entering certain things. When an error is identified we will go to that class and identify what is wrong and fix it so that it works with the program. This is how we make sure that all of our classes are working in our code.