We put a high precedence in working out the order of communication and inheritance between classes in our program before we started development this ensured we had a clear idea of how our program would be created and ensure we did not make any major errors in the design of our code.

The first class we looking into was the Player class this is a very important class as it defines the player and the way that we designed it. The player class is also the way we store the information for the player for example the Player class has the list of current toys and food of the player and even their pets are stored in this class there are a lot of important methods in player that we use throughout the program. The player class has to communicate with a variety of classes but most of all the food, toys and pet as the player holds the toys and implements a few methods from the toy class this is the same with toys and pet. There is no inheritance used with the player class as we decided it was unnecessary our original plan was to have a store class and have it inherit some of players methods but we later determined this not needed we use array lists that store type toy and type food and type pet to keep track of the player's inventory and pets..

There are Three notable abstract classes we used in our program which were the Toys class and the Pet class. We used theses so that their inheritance would be able to use their methods and change them as needed the inheritance for theses classes are the objects for example for Toys to name a few we have a BigBall and LongRope theses are toys that we use in our program that are made in another container called the ToyContainer each toy can vary in price and durability as the more durability the toy has the more it can be used without needing to buy another. The same for Pet class we have a some objects to name a few we have Burrito and Salmon and again theses are some of the food that we use in our program the food objects are stored in the FoodContainer each food can vary in nutritional value so with higher nutritional value the amount of hunger recovered will naturally increase.

There is also the Pet class which is not an abstract class as the methods in the Pet class are widely used within the program and Pet inherited the Species class which is used to define the various species that pets can be Species is also an abstract class as we use the methods for the species objects which are defined in the SpeciesContainer a few example of theses are the Manatee and the budgie the pet species can vary in not only name as the species have different weight and damage values.

Lastly there are two classes that are responsible for the GUI there're GUI Main and GUI Creation first Creation is called as it builds the game and takes player input after that it calls the GUI Main which handles the main game background environment and GUI for the game theses two classes interact heavily with the player and pet classes as Creation is creating the player and the players pets and then the Main is using that information to play the game and is continuing to interact with and update the pets and player each day.

Unit test coverage is the coverage of the test of the program more specifically if you have high coverage it will test more of the source code than low coverage.

We ran junit tests on our main classes more specifically on our Food, Pet, Player, Species and Toys classes. The Toys class did not need much coverage as most of it was just getting and setting values which was not needed to test this is the same for the Species and Food class which only had getters and setters but for the Toys class we did have to check one method which was removing durability we made sure to cover all possible input points to make sure than any possible errors in the method was found. The player and pet classes had a lot of methods that we needed to test we made sure that in each test we were testing a variety of the inputs so that we could cover as much possible errors a few errors did show up and we were quickly able to fix the problems .

The assignment was overall very good it was the right amount of challenging aspects and was the right difficulty in that we were still able to do the assignment without too much help but still challenging us. The main problems we felt were in time it would of been better if we had learned GUI a bit earlier in the course as we felt like we didn't have enough time to implement some extra features we had originally planned.

The program went quite well in itself and was quite smooth we didn't have a lot of bugs we had a few but were able to easily find and fix them. Time management would definitely be our downfall and the main improvement would definitely be more time management and to make a plan on when to work and meet up together to do work in person.

Jade Martin 50%

* Did a lot of the background coding for example planning out how to do the actions and interactions and implementing them and also a lot of the GUI.

Kyran Stagg 50%

* Did a lot of GUI, mainly creation and worked on bugs in GUI. Created UML and half Junit tests