Design Rationale

**Oil Can**

The Oil Can healing object that needs to be created will inherit from the SWEntity class in the Starwars entity package. This class will have dependencies which are the take and capability classes. The oil can can be picked up by all users of the application but can only be used on the droids to increase their hitpoints. A healing action class will also need to be created in the starwars.action package to deal with the healing that the canteen and the oil can will do. The healing action will simply just increase the users hit points by a certain amount (2). As the oil can is an infinite source for the droids, the healing from the oil can cannot outdo the damage of the attacks which will be set to a higher value than the healing.

**Force Ability**

We will be creating a force class that will be created in the starwars.actions class that will be instantiated in the SWactor class. All characters in the game will have various levels of force depending on the type of character they are. For example, the tusken raiders will be given a random force level between 0-2, the droids that will be created will have a force level of 0, Ben will be given a high force level of 10 and Luke will be given a low force level but this will be incremented throughout the duration of the game when he interacts with Ben in the training Luke class. Uncle Owen and Aunt Beru will be also be created as part of the specifications from Assignment with a force level of 0 which means they are able to be mind controlled.

**Luke (Player)**

Since Luke is already created in the game, he will gain other attributes such as the force ability that all other actors will have. Luke will be given an initial force ability value of 3 Luke is able to interact with Ben Kenobi to increase his force ability. Once they have interacted, Luke will be trained and his force ability will increased by 2 (incrementForce method). Once Luke has enough force ability (8), he will be able to use the lightsabre as a weapon instead of just being able to hold one.

**Droid**

A droid class will be created that will inherit from the SWActor class which will be made in the starwars.entities.actor. The droids will have a force ability instantiated to 0 as per the assignment requirements. There will be an arraylist in the SWActor class to hold the droid of each respective owner. Droids will be able to interact with most of the objects in the game but some objects they can directly interact with to change their attributes values. The oil can can directly interact with the droid to heal the droid. The droids will go to their owners position that they were on the turn before which will be kept in variable that saves the owners x and y position on the interface. Another function that we plan to implement that directly interact with the droids is a class to repair the droids using the parts from the immobilised droids. This will inherit from the SWActions class in the starwars.actions package that is already in the application.

**Lightsabre**

Since there is already a lightsabre class in the starwars.entities package, we will not need to create the class. Therefore the lightsabre is able to interact with all actors in different ways. They can be held by each of these actors but only those with a certain threshold of force ability will be able to wield the lightsabre as a weapon which will only be Ben and Luke who have the capabilities to have a high force ability. In order for the lightsabre to have the ability to attack other actors, we will also need to create a lightsabreattack class which will inherit from the attack class that is already in the starwars.actions package. This action will be dependent on the lightsabre as seen in the UML class diagrams.

**Mind control**

As part of the specifications, the actors with any amount of force (>0) are able to control the weak minded (0 force ability) characters. This class will be created in the starwars.actions package which will inherit from the already implemented SWAction class. This class will contain an array of weak minded characters that the actors with large amount of force are in control of. In this class, it will check the force ability levels of the actors in the action and decide on whether mind control can take place (An actor with force is trying to mind control another actor with no force). The mind control will be able to move for one turn based on the command of the user.

**Sequence Diagram**

For the sequence diagram that we created, we found that there were 2 different actors and 2 other classes involved in the interaction. We assumed that all of the decisions were a pass to show a better flow of actions. The first interaction between the classes is by the force actor. The force actor wants to use the mind control action that we will create so they must interact with the mind control class. In order for a user to be eligible to use mind control, they must have a certain threshold of force (5). The mind control class interacts with the force ability to get the force level of the actor and returns it back to mind control to check if the actor is eligible. After the actor check is done, we need to check the force level of the mind control target to see if they are eligible also to be mind controlled. The mind control target will interact with the force ability class to obtain the force level of the target. If the targets force level is zero then they will be eligible to be mind controlled by the original actor. Then the action of being mind controlled is underway and the target is in control of the actor.