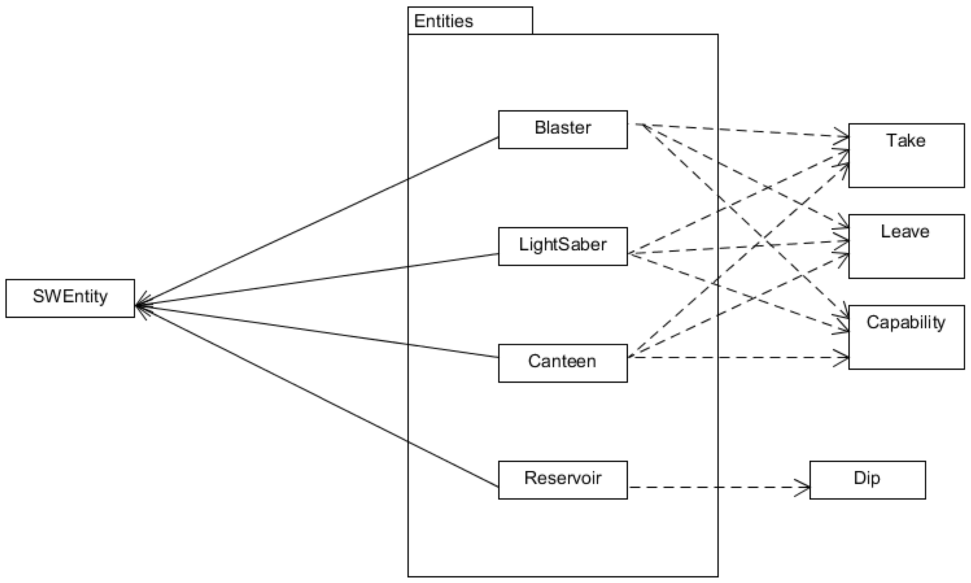
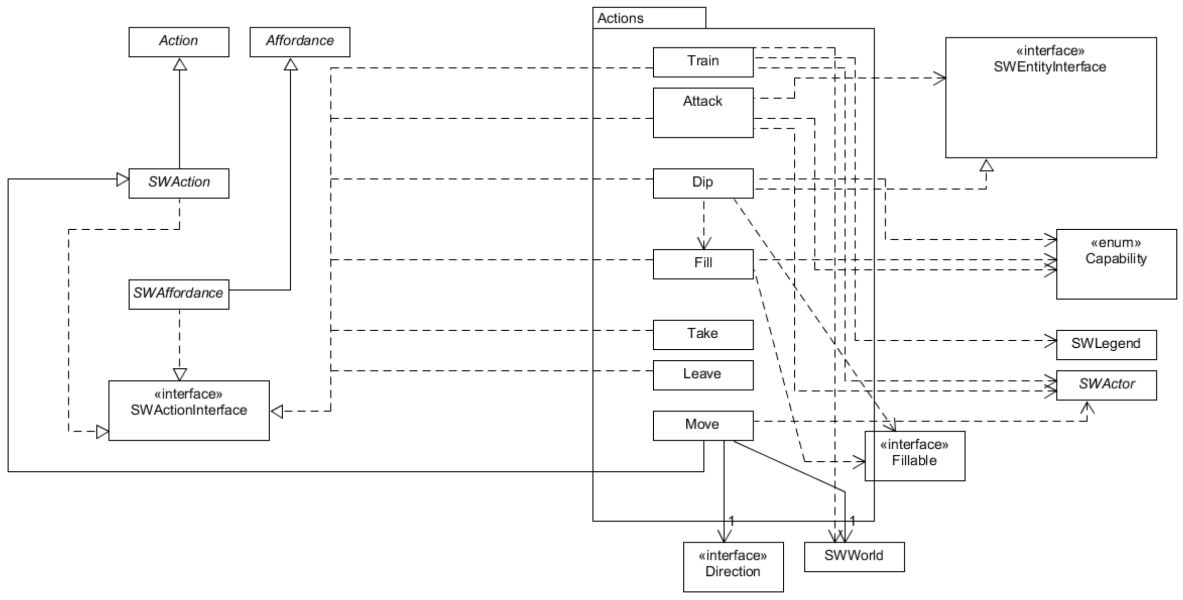
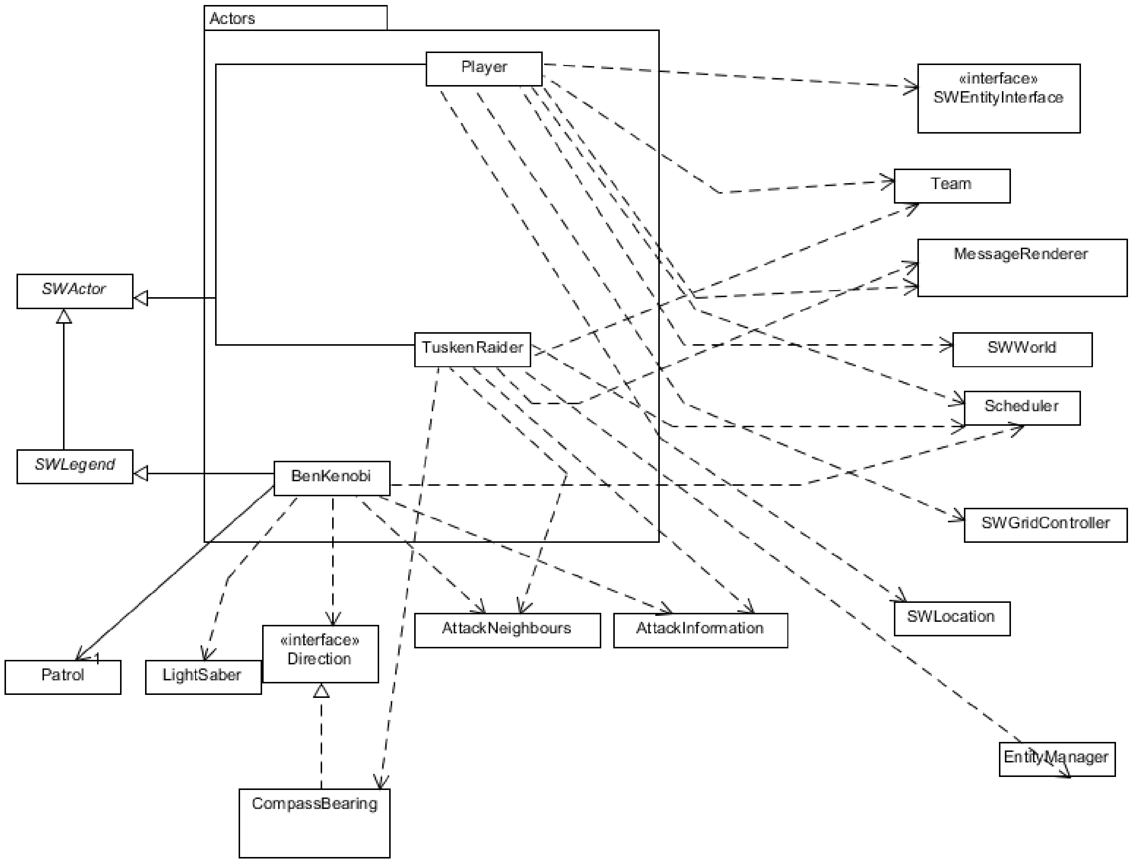
## 1. Leave Affordance:





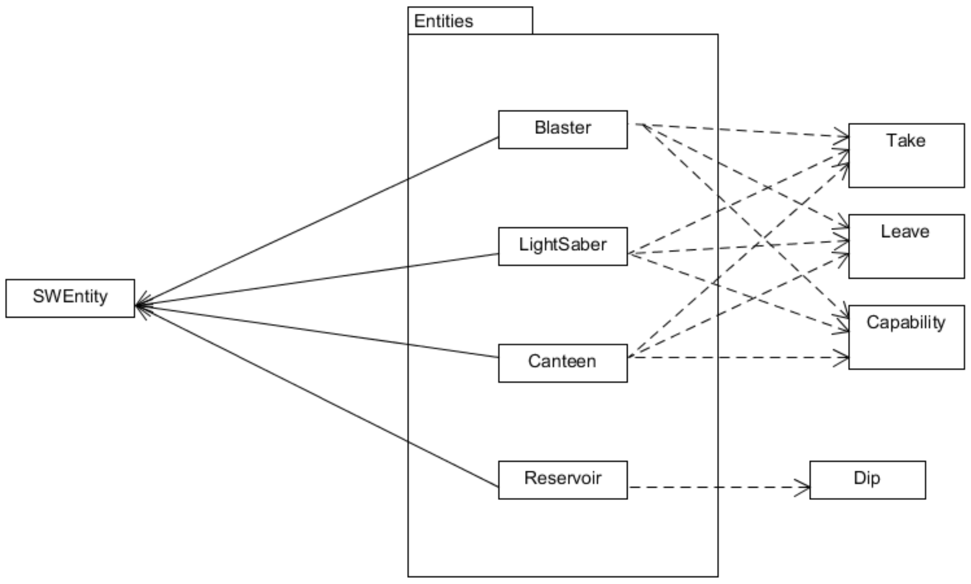
- Leave is a new class created in the starwars.actions package. It extends the SWAffordance class. The class contains the methods "canDo", "act". The method canDo checks if the actor is carrying an entity. The method act  checks if the target is an Entity. If so, it assigns it to a variable. Then we call the method "setItemCarried" of the actor and set it to Null. We then add the entity back to the map in the location the actor is currently in, and to the entity manager. We then remove the leave affordance and add the take affordance so that the item can be picked up again. The class also Has a description method that prints a description of the action.

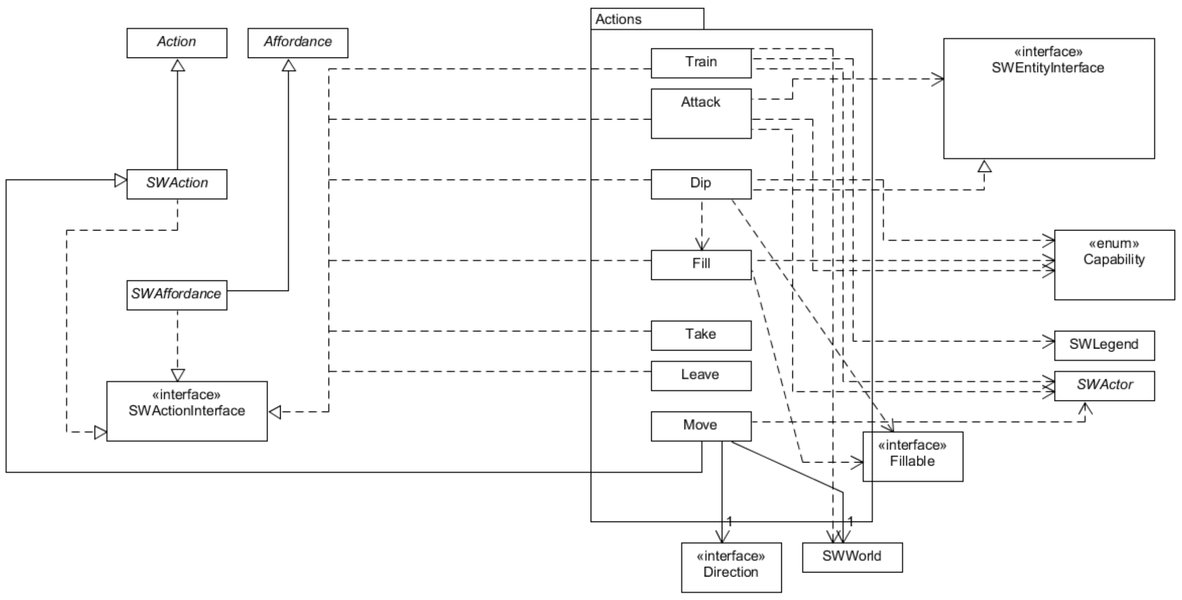
## 2. Force:



We added an integer attribute called force in the class SWActor. This attribute keeps track of the actor’s force level. If the force attribute is above a certain threshold, then the actor can use the force. We also added a setter and getter methods for retrieving and changing the value of an actor’s force.

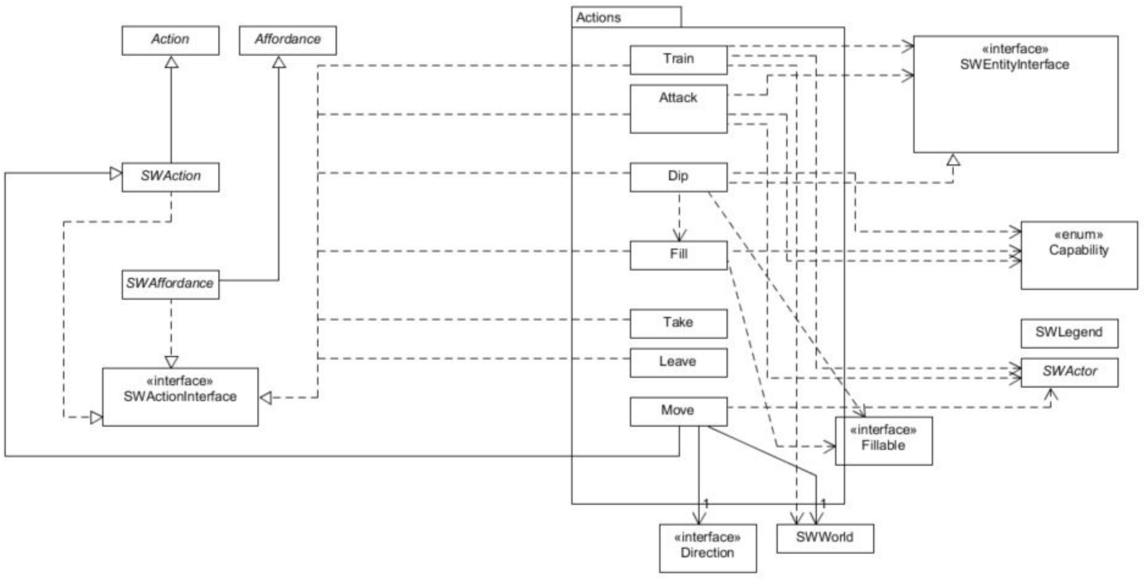
## 3. Lightsabres:





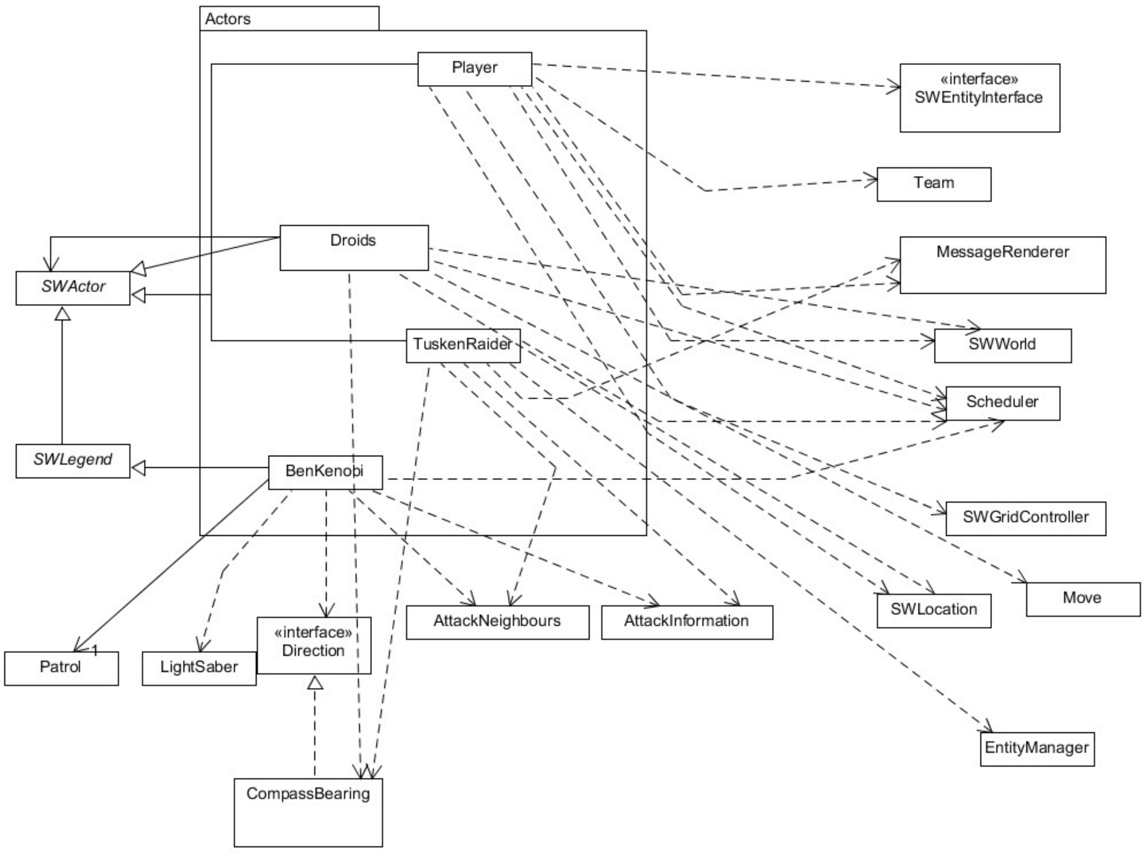
We modified the Attack class such that unless the actor is using a lightsaber and their force level is high enough, the maximum potential damage that can be dealt by the lightsabre is not dealt.

## 4. Ben Kenobi:

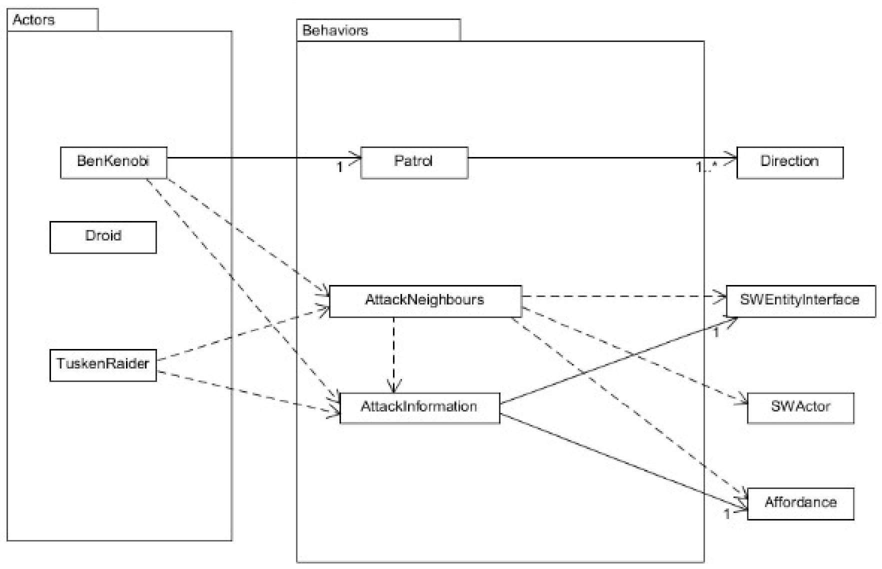


We created a new “action” class called Train in the action package. This class checks if Luke and Ben are in the same location of Luke and Ben using the method of Entitymanager in the SWWorld class (`SWWorld.getEntitymanager().whereIs()`). If both of them are in the same location and luke hasn't been trained by ben before, then it increases Luke’s force level so that he’s able to use lightsabres.

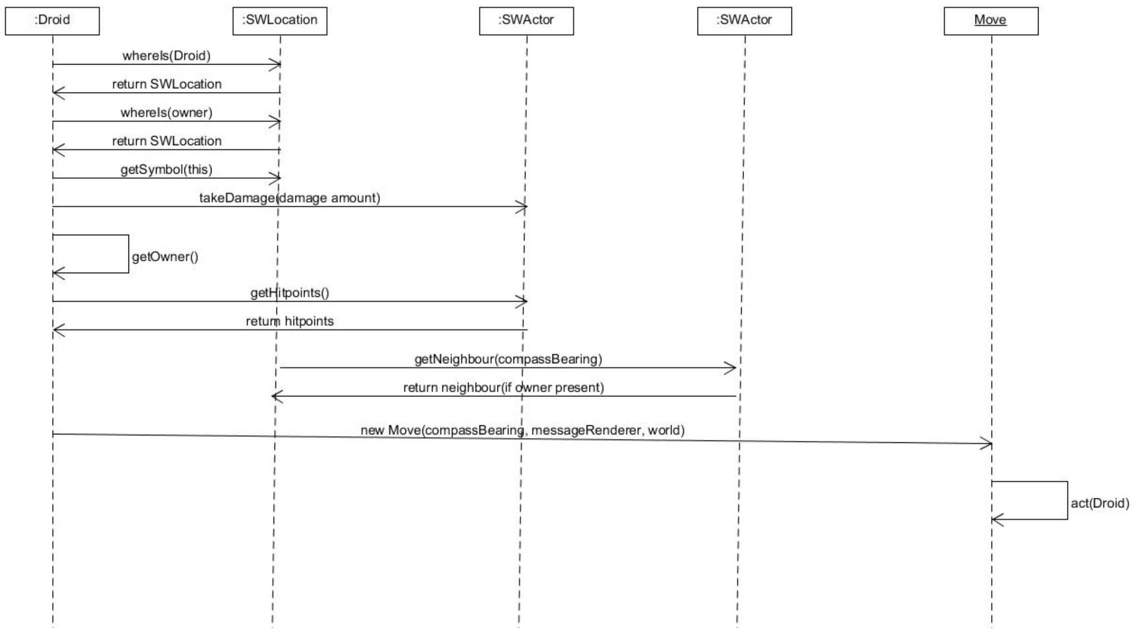
## 5. Droids:



We had to add some more relationships between the Droid class and some of the other classes because as we tried to implement the design, we realised that we had left out some important connections that were needed to make the code function as desired. For instance, we need the CompassBearing Class to form our list of possible movements when the droid needs to pick a random direction. We also need the Move class to make the Droid's location actually change the way it is supposed to. Finally, we needed the SWLocation class to find the owner and compare its location to that of the Droid (see if they are together), as well as to check if the owner is in a neighbouring location.



We decided to remove the FollowOwner Class from the behaviour package because we realised we could do it from within the Droid class, in the ***act()*** method that all SWActor objects have. In the ***act()*** method we take care of the motion of the droid depending on its position in relation to its owner if it has one.



This sequence diagram shows when the droid starts in the “badlands” and finds its owner in a neighbouring location. It will take damage because it is in the “badlands”. We must then see if it is still alive/ able to move after taking damage, if it is then we search the neighbouring locations for the owner and get the compass bearing, which we will use to do the move. We find the owner and create a new Move object. We then use the ***act()*** function in the move class to change the droids location. The sequence is now complete.