**Force Ability:**

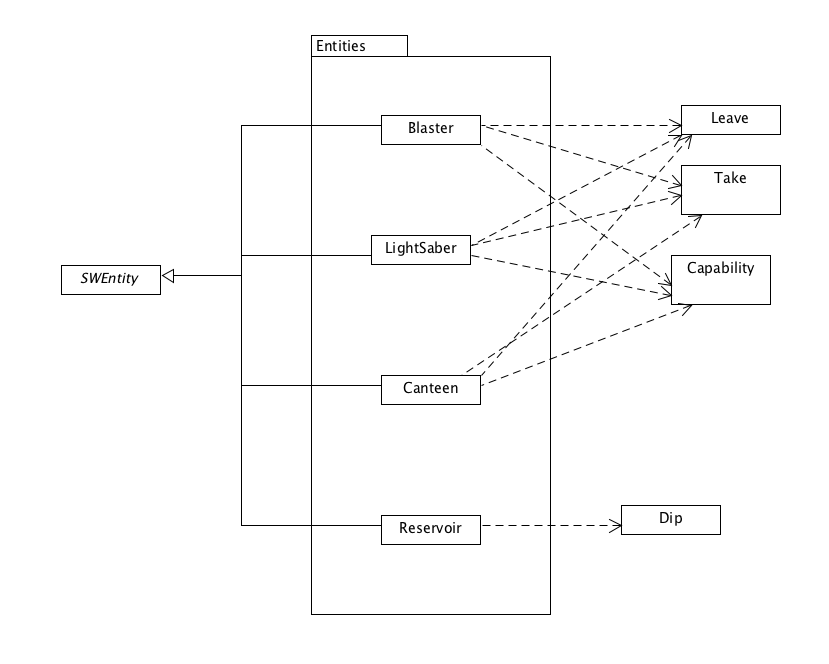
The force ability is a class that contains the force levels of each player(character). Different force levels come with different abilities the higher the level the more abilities the player can use. Some players can be trained to increase their force levels. Not all players have “The Force”.

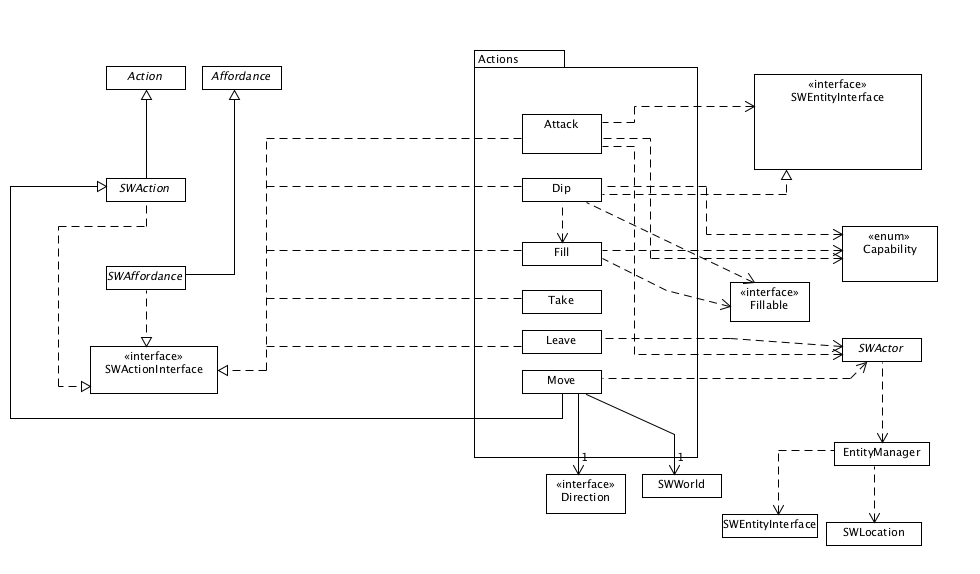
How it works:

The force levels start from 0 which means the player has no Force Ability

And goes all the way to level 10 which is the highest level.

**Leave Affordance**



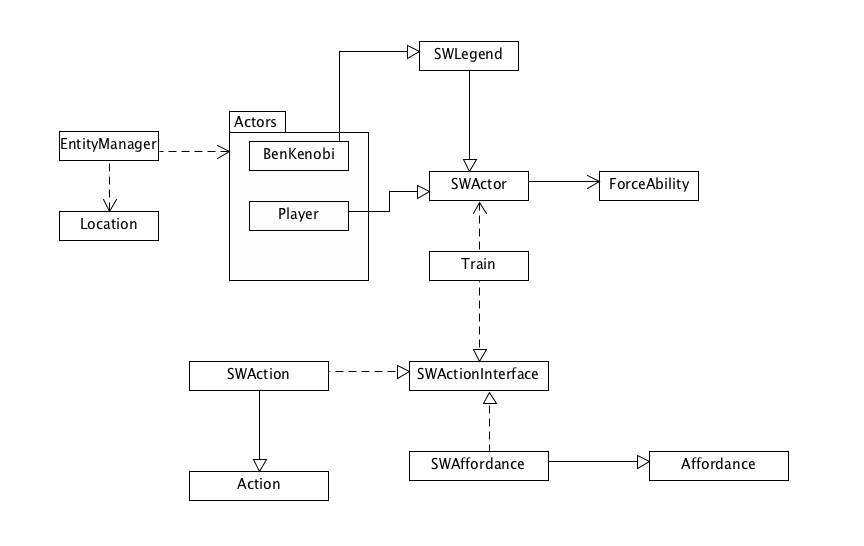


It is an action class which enables the player to discard the items they no longer need .

This class is connected to the location attribute of the player, so that the item can be left at the correct location which is the player’s current location.

How it works:

* Check if the player have an item
* Check if the item can be discarded
* It yes,
  + Check the current location of the player
  + Put the item at the current location by changing the SWGridContoller()

**Ben Kenobi:**

The modified functionality in Ben Kenobi class is to give the ability to train and raise the force ability of Luke. Ben and Luke should be at the same location.

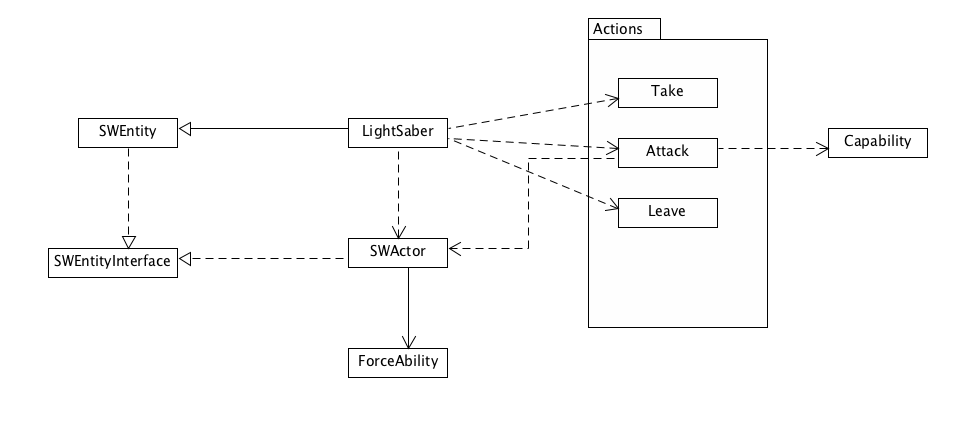
Design Rational:

* We decided to implement the train by extending the action class which can be used by SWActor so any player can have the ability to train if needed. However in this stage only Ben is implementing the train ability. Apart from that it will be easier to manipulate the train class if different kind of training to be provided later in the system.

How it works:

* If Luke and Ben at same location:
  + If Luke force ability == 10:
  + Yes, refrain from putting the option into the command
  + No, show command that allows Luke to be trained
    - Increase force ability of Luke by 2

**LightSaber:**



The lightSaber is an entity that can be picked up by any player but not every player can use it to attack.

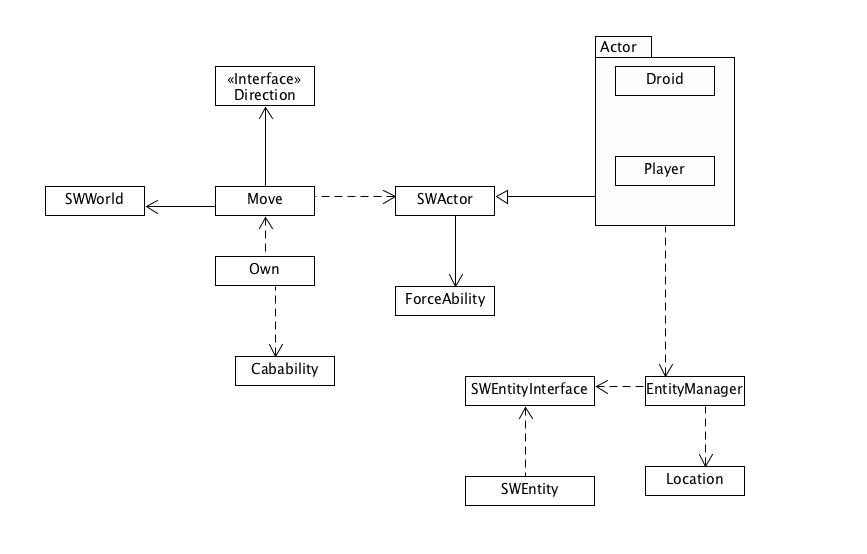
Players with force ability more than 6 can use the lightsaber to attack.

Design Rational:

* We decided to add dependency of attack on SWActor so that we can check whether a given actor has the enough amount of force ability to wield a lightsaber

How it works:

* Actor picks up the lightsaber
* If actor’s force ability >= 6:
  + Yes, attack ability is enabled for that actor
  + No, attack ability is disabled for that actor

**Droids:**

Design Rationale:

The droid class we are implementing is an actor class with a unique set of actions.

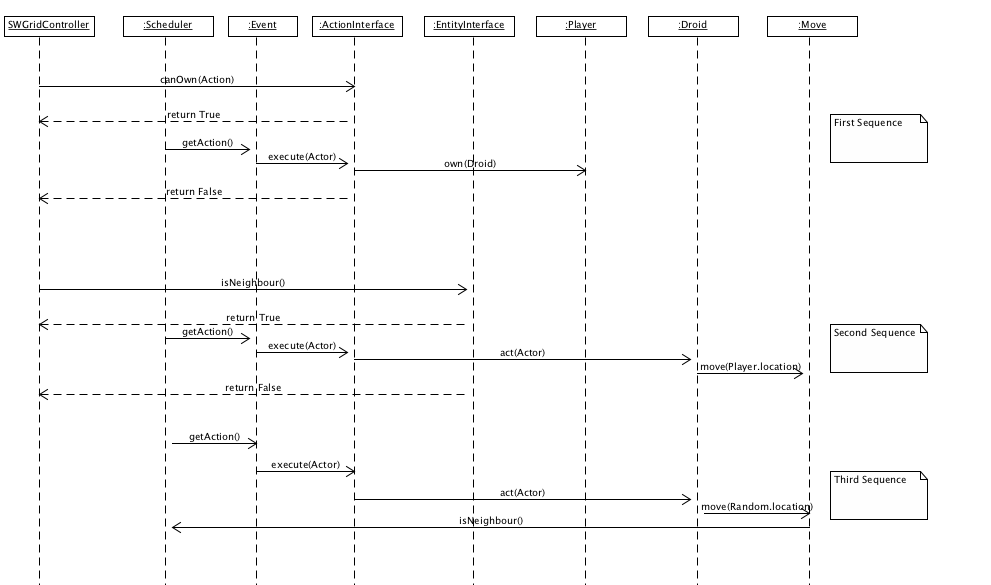
A Droid cannot move from its place unless it's owned by a player. A droid does not have any force ability. A Droid’s movement always depends on its owner location. A Droid’s location can have an affect on its health

How it functions:

* If a player is in the same location as a droid
  + Yes, if the droid does not have an owner
    - Yes, player can own the droid
* If a player’s droid location is at a neighbouring location
  + Yes, move the droid to the player location
  + No, move the droid to a random location
    - Keep moving in that same direction until it finds its owner
    - cannot move in the same direction
      * Yes, change to another random direction
* If the droid is travelling traversing through the badlands it loses health
* If the droid’s health runs out:
  + Droids can not move any more

*Note: refer to the sequence diagram for further details.*

**Sequence Diagram for Droid:**



**First sequence:**

It checks whether the Droid is owned or not if true it moves onto the next sequence if it is not owned then the player can choose to own it if and only if the player and the droid are in the same location

**Second sequence:**

This sequence checks if the owner and the Droid are in neighboring locations. If yes, the Droid should go to its owner’s position if it's able to (still have health points).

**Third sequence:**

This sequence takes place when the Droid and the owner are not in neighboring locations if they are not in neighboring locations the Droid makes a random move then checks if they are neighbors if yes it performs the second sequence if not it keeps moving in the same direction checking if they are neighbors with every move until it's unable to move in that direction then it chooses a new random direction and performs this sequence again.