**Referring to our assignment 1 documentations, the ones in blue are the changes we made in our design, there are quite a few major changes made.

Starwars.entities.actors package

Class name: SWActor

Changes made:

We decided to introduce a new attribute into SWActor called ForceAbilityLevel which takes an integer of range 0-10 to fulfil part of the first project requirement (ForceAbility). (Similar to the hitpoint attribute.)

We have fulfil this part exactly.

Class name: SWRobots

Responsibilities:

- This class represents "robots" – all non-people characters.
- We decided to create a new class for robots because in the future, there should not only be one droid, there would be many different types of droids with different characteristic and name. Thus, with this new class, we could initialize all those characteristic by inheriting the SWActor properties. The functionality of this class has a little similarity with SWLegend.

We created a SWRobot class which extends SWActor. Instead of being similar to SWLegend, I find this class more similar to SWActor, for example: SWActor is responsible to check if an Actor is an owner to a droid or it has not own any droids yet, whereas SWRobot is responsible to check whether a droid is owned by any actor or not. This is why:

- in the SWActor class, we have the method setNotOwner() and setDroidOwner(). The attribute "droidOwned" will be null if the actor is not the owner of any droids. A droid object will be assigned to the "droidOwner" attribute if the actor owns a droid.
- In the SWRobot class, we have the Boolean attributes hasOwner which will be true when method isOwned() is called and the attribute will be assigned "false" when the disowned() method is executed.

Class name: Droid

Resposibility:

- cannot use force --> force level = 0

- Owned by an actor. All droids have an Own affordance in which it can be owned by any SWActor. Once it is owned by an actor, it's Own affordance will be removed and replaced by Un-own affordance which means the droid can be disowned whenever the actor feels like disowning the droid. As long as the droid is owned by an actor, it's characteristics will be following the owner's characteristic (eg Team and movements)

- As long as it's hitpoint is more than 0, the droid will follow the owner:

  droid.schedule = owner.schedule- 1

  ** because the droid has to be one step behind of the owner.*
  ** this is part of the reason why there is no relationship between Droid and Scheduler class*
  **Another reason is because once the Droid is disowned/ when it's hitpoints becomes 0, the droid will be immobile. It will remain at the same point and wait for another owner or some other actor to heal it.*

#There is some misunderstanding did in Assignment1, we didn't fully understand what does the scheduler does, thus this point is totally unusable for our design. Thus we implemented some other design.
When an actor(Luke) meets a droid, there will be an option for the player to own the droid if the droid is not already owned by some other actor. This is made possible because the Owned affordance is added to every SWRobots in the SWRobots class.

- *Once the droid is owned, the Owned affordance will be removed from the droid and the Disowned affordance will be added. All of this is done in the setDroidOwned method in the SWActor class. But sadly I don't know what went wrong, when the droid is disowned, it cannot patrol back anymore. This is the bug that I am still trying to fix.*

- loses hitpoints when move in Badlands. There will be a method that checks the location of the droid and reduces it's hitpoints if it's at Badlands (class collaborator: SWLocation) There is dependency between Droid and SWLocation because the owner might be in Badlands but the droid is one step behind, thus the droid's hitpoints

#Again there is a misconception about the droid have to be one step behind of the actor instead of being in the same coordinate. For this point, we check for the location by using the entityManager.whereIs(this) then get a description of that place and if it's Badlands, the droids' hitpoints will reduce.

will not be reduce. Thus, we should identify the location of the droid itself instead referring to the owner's location.

- Droids are given the Repair affordance hence they can be repaired when they are disassembled → only when they are immobile that they can be disassembled.

Class name: DroidParts

Responsibilities:

- DroidParts is an entity. The dependency between Droid and DroidParts is similar to the dependency between BenKenobi and the LightSaber. All droid will be carrying a few DroidParts, but all those parts will only be available to be taken if and only if the droid is immobile.
- More detailed explanation on this class is made in the starwars.entities package section.

Starwars.actions package

Class name: SWRobot

Responsibilities:

As explained in the starwars.entities.actor package section.

Class name: Drink

Responsibilities:

- An affordance for SWActor to drink from DRINKABLE entities.
- The reason there exist dependency between Drink class and Heal class is because right after SWActors drink any liquid (including droids can drink oilcan themselves to regain health/ heal) they would regain hitpoints or in another word they will heal. This dependency is similar to the dependency between "Dip" class and "Fill" class.
- This class make sure that the SWActor to be healed have not reach its maximum hitpoint and the entity with the HEALER capability has enough hitpoints before healing the target.

Class name: Heal

Responsibilities:

- Similar to Attack action, but instead of causing damage to another entity, this class heals entities by increasing their hitpoints.
- It will increase the hitpoints of the entity to be healed and blunts the DRINKABLE used for the heal (reduces the hitpoints of items used to heal -- with DRINKABLE capability.)
- make sure that the person to be healed do not have maximum hitpoint and the HEALER entity has enough hitpoints before healing the target.
- For droids to heal themselves, they would DRINK from the oilcan. For somebody else to use an oil can on the droid to heal the droid, they would immediately access this class instead of passing through the Drink class. However, despite having to heal from drinking or from being healed by someone else, both actions will reduce/deplete the entity used to heal. This is why both Drink class and Heal class has to implement Drinkable interface to reduce/deplete the level of the entity.

#There is a slight major change for both classes. We combined both of this class into one action class named "Healing". We assume that whatever that we drink, it will help healing. This design might not be that ideal, because there might be some poison drink implemented which mean drinking does not necessarily heal. But if that's the case, we could add another capability to that entity poison drink lets call it POISON. Then we will have some if-else condition in healing class.

- For this healing class, we make use of the SWActor's takeDamage method to increase hitpoints, we deleted the assertion and input a negative number as parameter.

Starwars.actions package

Class name: Repair

Responsibilities:

- "revive" the SWRobot (eg. Droid) when the appropriate amount of DroidParts repaired has been reached.
  * Assuming that for a droid to be fully repaired and become "alive", it needs 5 droid parts before it is ready to be owned by other SWActor. This class checks the number of DroidParts that has already been "repaired" to the droid.
  * As the last piece of DroidParts is added to the Droid, the droid will be alive, however, it's hitpoints will not be full. It's hitpoint will start from 10. From there, the owner has to heal the droid using OilCan
  * A Droid has a Repair affordance doesn't mean the droid can repair itself, it means that it can be repaired.
- There exists dependency between Repair class and SWRobot class instead of SWActor class. This is because not all actors can be repaired, only robots can be repaired.
- We can only repair the robot if and only if the robot is disassembled and immobile. Thus, the dependency between Repair class and SWRobot class is needed as many information about the robot is needed. (conditions coding wise)

<<interface>> Drinkable

- Interface for SWEntities drinkable of water.
- All drinkable objects must have capability DRINKABLE.
- Contains a method which empties the SWEntities which previously is filled with water. This is similar to the <<interface>> Fillable, instead to having a fill () method in the interface, we would have an useUp () method. This method would exist in the SWEntities that had the DRINKABLE capabilities and would either reduce the amount of liquid in the entity (eg oil can) or empty out the liquid (eg canteen). → so instead of level = capacity, we would code something like level = 0 or level=level-amountUsed.

  #Previously, I mentioned that we have combine the Drink and Heal class into one Healing class, thus this Drinkable interface is already not needed.

Starwars.actions package

Class name: Train

Responsibilities:

- An affordance for Luke(Player) to increase his force ability and to be trained by actors with TRAINER capabilities.
- This class is similar to Attack action, but instead of causing damage to another entity, this class increases Luke's force ability level. There is a dependency between Train class and Player class because for this assignment, only the Player(Luke) can be trained by BenKenobi. But in the future, the Player might be able to be trained by some other SWActors as long as they have the TRAINER capability. Thus, for this assignment, only BenKenobi has the TRAINER capability which is to train the Player(Luke).

we did not add any new TRAINER capabilities. Because we thought it was unnecessary. Instead, we designed our game so that anyone with a higher forceAbilityLevel will be able to train any actor with a lower forceAbilityLevel. Assuming Ben has a forceAbilityLevel of 8 and luke has a forceAbilityLevel of 3, in this case, Ben can train luke until Luke reaches a level of 8 and no further. If Luke wishes to train to level 10, then he has to look for another actor with forceAbilityLevel of 10 and hope that that actor would be willing to train Luke.

Class name: MindControl

An affordance for SWActors to control the weak minded

Responsibilities:

- There will be a new method in the SWEntityInterface which is getForceAbilityLevel(). This is to check whether the mind control target can resist the action or not. If the target's ForceAbilityLevel is less than 4, then the program will proceed with allowing user to input the coordinates for which the target should be moved. Else if the target's ForceAbilityLevel is more than or equal 4, this method will return False and ask the user to choose another target with weaker ForceAbilityLevel.

*in another word, only SWActors with ForceAbilityLevel will have the MindControl affordance. An actor with MindControl affordance does not mean the actor can perform MindControl, it means that they can be mind-controlled

- Only SWActors with ForceAbilityLevel of 7 or above will be able to perform this mind-control action towards those with MindControl Affordance.

#Above implementation is implemented successfully. We did created a new MindControl class and a new method of getForceAbilityLevel() → renamed to getForce(). In this game, **we assume that every actor has the MindControl affordance, which means everyone can be mind controlled**, it's just that whether do u have enough forceAbilityLevel to resist it. Since mind controlling means the player get to decide what the other actor that is being mind control does, all the mind controlling act is in the Player class because userInput is needed and we need to getUserdecision which is all in the player class. One can only mind control actors in the same coordinate.

Class name: Own

SWAction that allows a SWActor to own a droid/robot.

Responsibilities:

- This class is similar to the Take class. The only difference is their target and the person who perform this action.
- This class has dependency with the SWRobot class and SWActor class. This is because this action can only be performed on robots/droids. In another context, only robots/droids can have the Own affordance. While the person who perform this action will be an actor (excluding SWRobots) because robots cannot own robots.
- As soon as the Own action is performed, this class will remove the Own affordance from the robot/droid and replace it with Un-own affordance which means the droid can be disowned whenever the actor feels like disowning the droid.

We implemented this class with a slightly different approach. However, the concept is somewhat similar. Most of the above implementations is executed in the SWActor class, this is because, when a droid is owned by an actor, it's schedule follow's the actor's schedule. Thus to have access to wards the actor's schedule, we find the most convenient way to do so is to implement above implementations in the SWActor class, under the setDroidOwned() method. The entity droid is passed in as parameter as well as the affordance Owned itself, so that we could remove that affordance from that entity and add the Disown affordance.

Starwars.actions package

Class name: Un-own

SWAction that allows a SWActor to remove ownership of a droid/robot.

Responsibilities:

- This class is similar to the Leave class. The only difference is their target and the person who perform this action.
- This class has dependency with the SWRobot class and SWActor class. This is because this action can only be performed on robots/droids. In another context, only robots/droids can have the Un-own affordance. While the person who perform this action will be an actor (excluding SWRobots) because robots cannot Un-own robots. (Since robots cannot own robots in the first place.)
- As soon as the Un-own action is performed, this class will remove the Un-own affordance from the robot/droid and replace it with Own affordance which means the droid can now be owned by any other SWActors.

For this class, we renamed the class as Disowned instead of Un-Own. Whatever that is described above is implemented in out Disowned class.

Starwars.entities package

Class name: LightSaber

Responsibilities:

- only people with ForceAbilityLevel of 7 or above can use this as a weapon.
- In the current code, anyone can wield LightSaber as a weapon, to apply the condition above, we would need to make some changes to the constructor of this class.
  - Create a new method which would check the actor's ForceAbiliyLevel after the actor take the LightSaber. If the ForceAbilityLevel is 7 or above, then in the constructor, we would add the Weapon capability to LightSaber, else do nothing.
  - This means that if the actor that picks up the LightSaber has a ForceLevelAbility of below 7, then the LightSaber will not have the Weapon capability.
- The dependency between LightSaber and SWActor is for us to get the Actor's ForceAbilityLevel.

#This requirement is done in the attack class. In the attack class, we would check the Actor's weapon and it's forceAbilityLevel first before allow them to attack, if the forceAbilityLevel does not reach the requirement, then we would suggest the player to leave the lightSaber and attack with barehands instead. Instead of restricting only people with forceAbilityLevel of 7 can use the lightSaber, we change the condition to only people with strong force can use lightSaber to attack. This means that actors that are weakminded and entities with no force will not be able to use it as weapon.

- There is a method in SWActor which checks whether that actor is weak minded or not. → isWeakMinded().

Class name: DroidParts

Responsibilities:

- Like the Reservoir class, DroidParts does not have hitpoints. We are assuming that DroidParts entity is immortal.
- everyone can take a DroidParts entity but not everyone can use that DroidParts to repair an immobile DroidParts. Therefore, there is a relationship between the DroidPart class and the Take class. To allow people to take the DroidParts.
- has the MACHINE capability.

Class name: OilCan

Responsibilities:

- "revive" the SWRobot (eg. Droid) when the appropriate amount of DroidParts repaired has been reached.
  * Assuming that for a droid to be fully repaired and become "alive", ready to be owned by other SWActor, this class checks the number of DroidParts that has already been "repaired" to the droid.
  * As the last piece of DroidParts is added to the Droid, the droid will be alive, however, it's hitpoints will not be full. It's hitpoint will start from 10.

#We did not create another new class as OilCan for this assignment because it seems to be like only R2D2 will be using that OilCan, thus, we assigned a canteen to R2D2 instead. As the functionality is similar whereby consuming/using that entity results in healing. In future , if more robots were to use the oilcan and there is a distinct difference in the property of the oilcan and canteen then an oilcan class will be created in the entity package. This class will be similar to the canteen class.

<<enum>> Capability

New capability added: “MACHINE”

Responsibilities:

- This MACHINE capability allows an entity to repair another entity which has the Repair Affordance
- Similar to LightSabre, everyone can pick a DroidPart up, but only people with the ability to repair can use this DroidPart entity and use it as a MACHINE to repair a droid.

New capability added: “HEALER”

Responsibilities:

- This HEALER capability allows an entity to heal another entity which has the heal affordance.
- Similar to the WEAPON capability (which allow an entity to Attack another entity which has the Attack Affordance) instead of causing damage on the other entity, heal affordance increases the entities hitpoints.

#We did not add a new capability because we thought it was redundant for now. This is because we assume that everything DRINKABLE will be healing. Thus, if in the future, a poison drink is introduced then we would need to add this healer capability. But for now, I don’t see the necessity to include it.