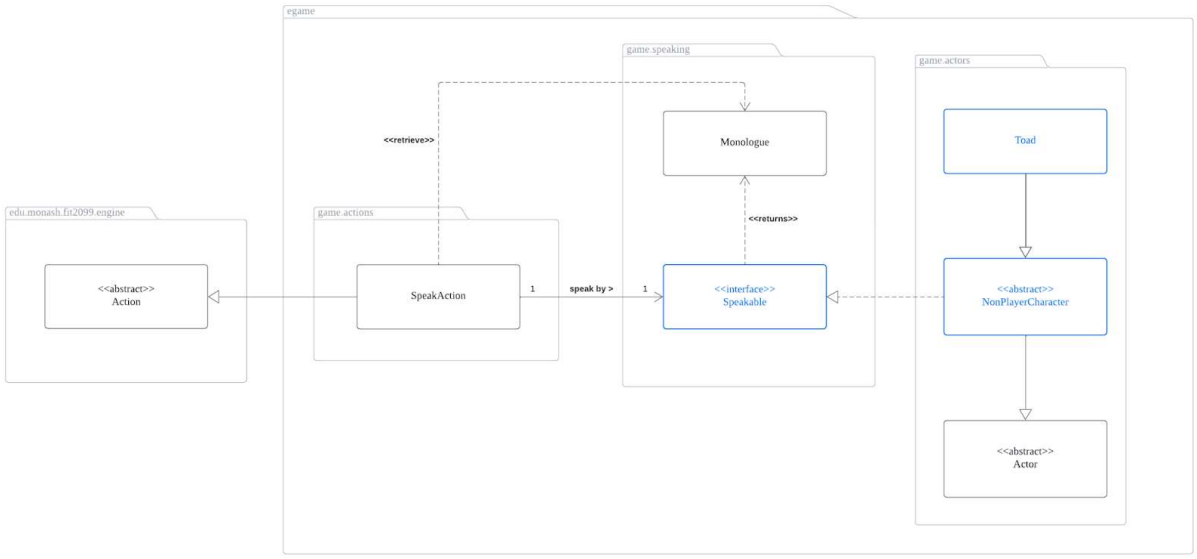
REQ6

Class Diagram:



Requirement:

- The player can speak with Toad if Toad is in its range
- Toad must not speak certain sentences in certain scenarios.
- Toad must randomly choose one of the sentences from all the available sentences

Design Rationale:

Firstly, to start this requirement, we have created a **Toad** class which extends Actor abstract class as Toad is classified as an actor. We initialise the instance of Toad with 3 input parameters, which is its name (Toad), its displayChar (0), and its hitPoints which theoretically can be any number that are greater than 0 since any of the actor in the map will not be able to hit the Toad. We would also override the `allowableActions()` method such that when the **Player** is getting the allowable actions from the Toad, it will return a **SpeakAction** action as one of the allowable actions that can be performed between the Toad and the player.

Besides, to ensure that no other actor can hurt the Toad, we will not return AttackAction as one of the allowable actions that can be done on Toad. Basically, the

allowable actions that can be done on the Toad by the player are just the **SpeakAction** and the **TradeAction,** and any other actor would not have any allowable action that can be done on the Toad.

In order to handle which sentence is possible for the speaker to speak, we have created a new interface called **Speakable** interface. The interface will be implemented by the **NonPlayerCharacter** abstract class which is the subclass of Toad since all the NPC can speak right now. For the current requirement, Toad needs to override the generateMonologue() method that are the abstract method in the **Speakable** interface. In this method, we will create a local variable of ArrayList<Monologue> and add the new instance of **Monologue** into the ArrayList<Monologue> such that each **Monologue** objects represents a speakable sentence. For instance, there are 4 possible monologue that can be spoken by Toad, thus we will add 4 newly created and initialized **Monologue** objects into the ArrayList<Monologue>.

To elaborate more on filtration of sentences according to the scenario, we will first explain on the **Monologue** class. As mentioned before, each **Monologue** object will represents a sentence, hence, the Monologue class will have 3 attributes,

1. (String) sentence: the sentence that can be spoken by the speker
2. (Speakable) speaker: the speaker that speak the sentence
3. (boolean) isAvailable: is the sentence available for the speaker to speak

In Toad's generateMonologue(Actor actor) method, we can done the checking by,

1. Initialized two boolean variables, hasWrench and isInvincible to false
2. Since all the actor can self-speaking according to assignment 3 REQ5, we will check if the target is not equal to itself.
3. If false, we will proceed to the Monologue adding parts mentioned above since there is no special condition that the speaker would not speak any of the sentences. And all the Monologue objects isAvailable attribute will be initialized to true
4. if true, means the speaker is talking to another actor. Then we will need to check whether the target (player in this case) holding wrench or is invincible and saved the result to hasWrench and isInvincible attributes.
5. Then, we will add initialized new Monologue objects and add them to the ArrayList<Monologue>

```
sentences.add(new Monologue("You might need a wrench to smash Koopa's hard shells.", this, !hasWrench));
sentences.add(new Monologue("You better get back to finding the Power Stars.", this, !isInvincible));
sentences.add(new Monologue("The Princess is depending on you! You are our only hope.", this, true));
sentences.add(new Monologue("Being imprisoned in these walls can drive a fungus crazy :(", this, true));
```

The isAvailable attribute in Monologue object will be used in filtering the sentences in SpeakAction `execute()` method.

Next, as we mentioned earlier, one of the allowable actions will be returned to the player when he/she is in range of the Toad will be new SpeakAction(this). **SpeakAction** is a subclass of Action which we created to handle the conversation, including the conversation between the player and the Toad. SpeakAction class has an attribute speaker which is data type of Speakable which indicates which actor is the one that is responsible to generate the monologues and speak. The implementation of this SpeakAction class follows the **Single Responsibility Principle (SRP)** as this class will only have one responsibility, which is to handle the retrieval of possible Monologue from the speaker which is Toad in this case and randomly choose the Monologue and returns the sentence as a String. The **SpeakAction** class follows the **SRP** since it is just a class that retrieve the possible monologues and randomly chooses one of them to return. Basically, just like the name of SpeakAction class, it just handles what to speak and speak out the monologue chosen.

We would override the `execute()` method in the **SpeakAction** class to create a new ArrayList of data type Monologue - availableMonologues. We will get the possible monologue by calling the generateMonologue() method of the speaker and check whether it is available, if true, then we will add it to the availableMonologues. After that, we will just randomly choose the sentence by using `availableMonologues.get(rand.nextInt(availableMonologues.size()))` and return it.

The design above that involved Speakable interface, SpeakAction class, Monologue class follows the **open-closed principle (OCP)** since it is open for extension and closed for modification. This is because the new actor that can speak will just need to

implement the **Speakable** interface and override the necessary method. The design of Speakable interface also fulfil the **Interface Segregation Principle (ISP)** since the interface is small enough and only implemeted by the classes that can speak. Hence, the system is more extensible and the classes that implement this interface can fully substitute the interface.

*Updates that have been done for REQ6 (class diagram & design rationale)

I. Class diagram:
   A. Added in a new **Speakable** interface
   B. Change the code implementation of the **Monologue** class and **SpeakAction** class
   C. Added in **NonPlayerCharacter** abstract class that extends by Toad

II. Design Rationale:
   A. Reimplement the handling of conversation between Toad and Player
      1. Older version:
         a) Hard coded the **Monologue** class to store and do the filtering of possible sentences between **Toad** and **Player**
         b) **SpeakAction** only acts as a middle man and call the **Monologue** class to do the necessary actions which violates the design smell of middle man.
      2. Newer version:
         a) Each instance of **Monologue** class now represents a single monologue sentence
         b) **SpeakAction** will in charge of choosing one of the sentences from the possible sentences that can be spoken and return it
         c) `generateMonologue()` method in the **Speakable** interface will be override by all the classes that implement it.
         d) `generateMonologue()` method will be used to filter and select possible sentences that can be spoken by the speaker