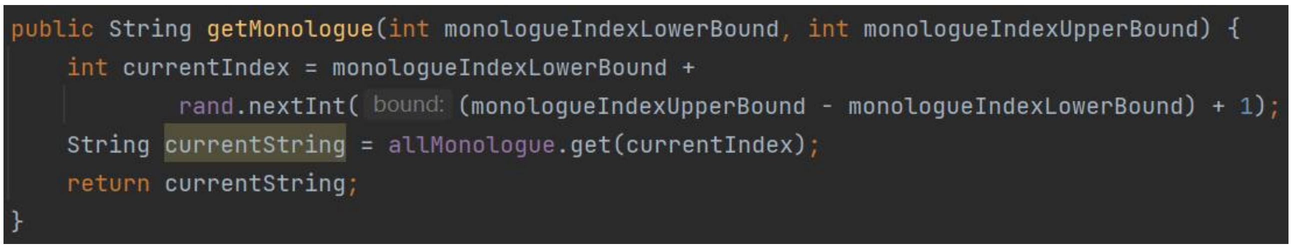
Assignment 3 REQ5: Speaking

Overview
To achieve this feature, there will be seven new classes (i.e., GeneralKoopa, FlyingKoopa, PrincessPeach, Bowser, PiranhaPlant, Monologue and Speakable) created in the extended system, and four existing classes (i.e., Enemy, Toad, Koopa and Goomba) will be modified. The design rationale for each new or modified class is shown on the following pages.

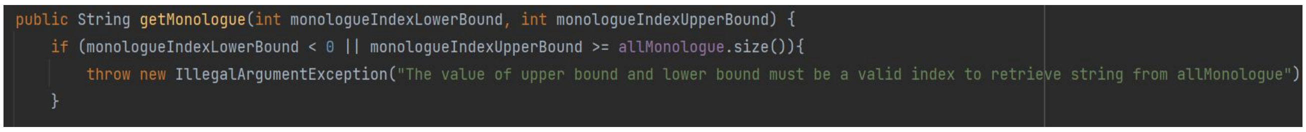
1) Monologue

Why I choose to do it that way:
According to the requirement states in Assignment 3 REQ5, I create a new class called Monologue to store all the monologues of actors. In this class, an arraylist is created to store all the monologues and a getInstance() method is there for us to get the instance of this class. Other than that, the way I make it to get the monologue from that arraylist is using the boundary index. In the getMonologue() method, we input two parameters: monologueIndexLowerBound and monologueIndexUpperBound. The integer currentIndex will be calculated using the boundary index, then the corresponding string will be return by arraylist.get(currentIndex).



```
public String getMonologue(int monologueIndexLowerBound, int monologueIndexUpperBound) {
    int currentIndex = monologueIndexLowerBound +
            rand.nextInt( bound: (monologueIndexUpperBound - monologueIndexLowerBound) + 1);
    String currentString = allMonologue.get(currentIndex);
    return currentString;
}
```

By doing this in this class, we don't need to repeat these methods in every actor class who is able to speak, so this obeys DRY(don't repeat yourself) principle. And if there is any new monologues added, we can add it into the arraylist directly and get it by index without adding monologue to each actor can speak, this make our code more tidy and logic. Besides that, Monologue class has its own responsibility which it performs the behavior of Monologue only, so it obeys SRP(single responsibility principle). In addition, I added exceptions for this method to make sure that the boundary index is valid. If monologueIndexLowerBound is less than 0 or monologueIndexUpperBound is greater than the size of the arraylist, an IllegalArgumentException will raise to notify the user the index is wrong. By doing this, I follow the Fail Fast principle that make our code more elaborate.

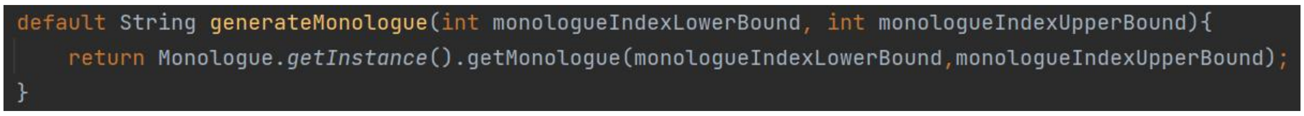


```
public String getMonologue(int monologueIndexLowerBound, int monologueIndexUpperBound) {
    if (monologueIndexLowerBound < 0 || monologueIndexUpperBound >= allMonologue.size()){
        throw new IllegalArgumentException("The value of upper bound and lower bound must be a valid index to retrieve string from allMonologue")
    }
```

2) Speakable

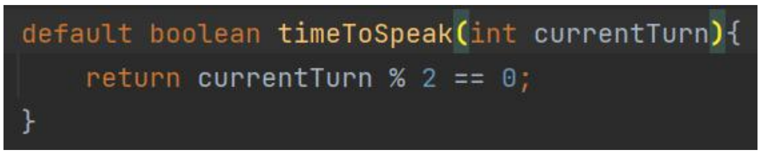
Why I choose to do it that way:

Speakable is an interface which contains two default methods: generateMonologue() and timeToSpeak(). In generateMonologue() method, we input the boundary index, it will return the monologue by get the instance of Monologue class and get the monologue from it by using the boundary index.



```
default String generateMonologue(int monologueIndexLowerBound, int monologueIndexUpperBound){
    return Monologue.getInstance().getMonologue(monologueIndexLowerBound,monologueIndexUpperBound);
}
```

As metionded in the Assignment REQ, each listed character will talk at every "even" or second turn (alternating). It means, they don't talk all the time. In timeToSpeak() method, we input the currentTurn counter, and divide the turn by 2, if the remainder is 0, that means the current turn is an even turn, which is the time to speak.



```
default boolean timeToSpeak(int currentTurn){
    return currentTurn % 2 == 0;
}
```

By doing so, we don’t need to repeat these methods in every actor class who is able to speak, so this obeys DRY(don’t repeat yourself) principle. Besides that, Speakable class has its own responsibility which it performs the behavior of Speakable only, so it obeys SRP(single responsibility principle). In addition, we followed Open and Close Principle(OCP) also as when there is more actor that can speak at even turn, we just need to let that actor class to implement speakable interface without modifying speakable interface code.

3) Enemy

What changed in the design rationale between Assignment 2 and Assignment 3 and Why:

According to the requirements state in Assignment 3 REQ5, all actors will speak every even turn automatically, so I make Enemy class implements Speakable, which an interface contains two default methods, one for generate monologues and another one is the turn counter. In the playTurn() method of this class, the currentTurn counter will increase by 1 in each turn of the game. Once the turn is even, a random monologue of particular actor will be generate and display. In the constructor of Enemy, I added the monologueIndexLowerBound and monologueIndexUpperBound, so all the class extends Enemy can input the boundary index directly and generate the monologues.

Why I choose to do it that way:

To reduce the repetitive code in those actor’s class, I created an interface Speakable and Monologue class to store all the monologues of this game. By implementing Speakable, we can use the default method inside directly without repeat the code in every actor who can speak, also every actor who extends Enemy can use those two methods directly without implement Speakable again, this obeys DRY(don't repeat yourself) principle. Besides that, Enemy class has its own responsibility which it performs the behavior of Enemy only, so it obeys SRP(single responsibility principle).

4) Toad

What changed in the design rationale between Assignment 2 and Assignment 3 and Why:
In Assignment 2, I added all the sentence belongs to Toad into the arraylist: toadTalk, and generate the monologue by using getReplyString() to modify which sentence to be talk depends on different capabilities the player had and different index to be used.
But In Assignment 3, the design is changed, all of the actors can talk every even turn automatically, so the way we get corresponding monologue is change to use Speakable interface and Monologue class instance.

Why I choose to do it that way:
To reduce the repetitive code in those actor's class, I created an interface Speakable and Monologue class to store all the monologues of this game. I let toad implements Speakable, so it can generate monologues by using the default method stated in Speakable and get the correct monologues by using boundary index. By doing so, we don't have to repeat all the methods in every actor's class, this obeys DRY principle.

5) PrincessPeach

Why I choose to do it that way:
By implementing Speakable, PrincessPeach class is able to use the default method stated in the interface Speakable. The speakable class will generate corresponding monologue by using boundary index. All the monologues are store in an arraylist in Monologue class, so we can get its instance and using the boundary index to retrieve the monologue we want. PrincessPeach class has its own responsibility which it performs the behavior of PrincessPeach only, so it obeys SRP(single responsibility principle).

6) Goomba

What changed in the design rationale between Assignment 2 and Assignment 3 and Why:
In Goomba class, I add the boundary index in its constructor, by doing so I can generate the monologues of Goomba directly since it is a class extends Enemy.

Why I choose to do it that way:

By doing so, the monologues of Goomba can be generate directly, if there is any new monologues added, we can modify the boundary index directly. Besides that, Goomba class has its own responsibility which it performs the behavior of Goomba only, so it obeys SRP(single responsibility principle).

7) GeneralKoopa

Why I choose to do it that way:
Because GeneralKoopa is a class that extends Enemy, so in the constructor of GeneralKoopa, the monologueIndexLowerBound and monologueIndexUpperBound is added, so all the class extends GeneralKoopa can input the boundary index directly and generate the monologues.

8) FlyingKoopa

Why I choose to do it that way:
In FlyingKoopa class, I add the boundary index in its constructor, by doing so I can generate the monologues of FlyingKoopa directly since it is a class extends GeneralKoopa. By doing so, the monologues of FlyingKoopa can be generate directly, if there is any new monologues added, we can modify the boundary index directly. Besides that, FlyingKoopa class has its own responsibility which it performs the behavior of FlyingKoopa only, so it obeys SRP(single responsibility principle).

9) PiranhaPlant

Why I choose to do it that way:
In PiranhaPlant class, I add the boundary index in its constructor, by doing so I can generate the monologues of PiranhaPlant directly since it is a class extends Enemy. By doing so, the monologues of PiranhaPlant can be generate directly, if there is any new monologues added, we can modify the boundary index directly. In the playTurn() method, I increase the currentTurn counter by 1 each trun of the game, when there is an even turn, the corresponding monologue of this class will display. Besides that, PiranhaPlant class has its own responsibility which it performs the behavior of PiranhaPlant only, so it obeys SRP(single responsibility principle).

10) Koopa

What changed in the design rationale between Assignment 2 and Assignment 3 and Why:
In Koopa class, I add the boundary index in its constructor, by doing so I can generate the

monologues of Koopa directly since it is a class extends GeneralKoopa.

Why I choose to do it that way:
By doing so, the monologues of Koopa can be generate directly, if there is any new monologues added, we can modify the boundary index directly. Besides that, Koopa class has its own responsibility which it performs the behavior of Koopa only, so it obeys SRP(single responsibility principle).

11) Bowser

Why I choose to do it that way:
In Bowser class, I add the boundary index in its constructor, by doing so I can generate the monologues of Bowser directly since it is a class extends Enemy. By doing so, the monologues of Bowser can be generate directly, if there is any new monologues added, we can modify the boundary index directly. In the playTurn() method, I increase the currentTurn counter by 1 each trun of the game, when there is an even turn, the corresponding monologue of this class will display. Besides that, Bowser class has its own responsibility which it performs the behavior of Bowser only, so it obeys SRP(single responsibility principle).