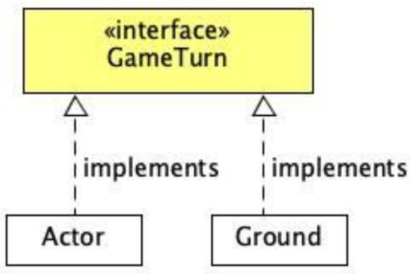
Recommendations to the game engine

Problem #1: Ground objects are not processed on every turn.

Problem explanation:
The World class in the engine package does not process every Ground object on the map in every turn. This is because only the Actor class contains a playTurn method. In order for a Ground object to be processed, an Actor has to be adjacent to it so that the allowableActions method is called. Since the allowableActions method is not a good indication of time, this makes it difficult to measure and make time pass for Ground objects. By relating this problem to our assignment, the issue we faced was with the oxygen dispenser. We could not leave the responsibility of scheduling the turns of dispensing the oxygen tank to the OxygenDispenser class (which is a subclass of Ground) as it is not processed on every turn unless an actor is adjacent to it for the allowableActions method to be called.

Proposed solution:



- An interface called GameTurn should be created. It has a method name called playTurn (similar method name and signature as the playTurn method in the Actor class) which returns an Action to perform.
- The Actor class is refactored to implement the GameTurn interface and overrides its playTurn method to provide its own implementation.
- The Ground class is refactored to implement the GameTurn interface and overrides its playTurn method to return null. Hence, the Ground class has its own playTurn method. Ground subclasses that needs to be processed on every turn to perform an Action will implement the GameTurn interface and return a suitable Action to perform in the playTurn method.
- The World class is refactored to have a method called processGround that iterates through every Ground object on the map and calls their playTurn method. If the playTurn method does not return null, the processGround method processes the Action by calling the Action's execute method. The processGround method is called in the World class' run method while its stillRunning method is true.

Advantage:
- Ground objects that perform Actions will be processed and executed. The Ground objects do not have to depend on an Actor's playTurn to be called on every turn to be processed.

- We suggested to create an interface, GameTurn to achieve abstraction so that the classes that implement it have to provide their own implementation of the methods in the interface. This increases loose coupling between classes.

Disadvantage:
- Since most Ground objects do not perform an Action, iterating through every Ground object on the GameMap reduces efficiency as it requires 2 for loops (to iterate through the height and width of the map).

Problem #2: Lack of documentation in the engine codes.

Problem explanation:
The classes in the engine package are not properly documented with javadocs. The lack of documentation makes it hard for us to understand the responsibility of the classes and its methods.

Proposed solution:
The responsibility of the classes and its methods in the engine codes should be properly documented by adhering to the Java coding standards. This includes:

1. The parameters of the method (@param)
2. The return values (@return)
3. Any exceptions the method may throw (@throws)
4. A thorough description on what the method does and how other developers could use it

Advantages of our proposed solution:
- Improves the readability of the methods.
- Allows developers who use the engine code to get a general idea of the method's responsibilities.

Disadvantages of our proposed solution:
- Takes time to manage and requires frequent updates when code functionality changes.

Problem #3: A WeaponItem object cannot use its superclass, Item's static newInventoryItem method without losing access to its own properties, ie. damage, verb.

Problem explanation:
The newly instantiated WeaponItem object cannot be added into the player's inventory right away as its superclass' static newInventoryItem method does not contain the parameters for int damage and String verb. Hence the WeaponItem object is upcasted and will only be treated as a normal Item since the Item class does not have the methods in its subclass, WeaponItem.

Proposed solution:

The WeaponItem class should have a static method newInventoryWeaponItem which has the same parameters as the WeaponItem class' constructor and in its method, clears its allowableActions and adds a newly instantiated DropItemAction so that it is added to the player's inventory immediately.

Advantages of our proposed solution:

- The WeaponItem object can be added into the Actor's inventory immediately.

Disadvantages of our proposed solution:

- There might be duplicated codes since the Item's newInventoryItem and the WeaponItem's newInventoryWeaponItem methods do the same thing.