Notes About Count.class

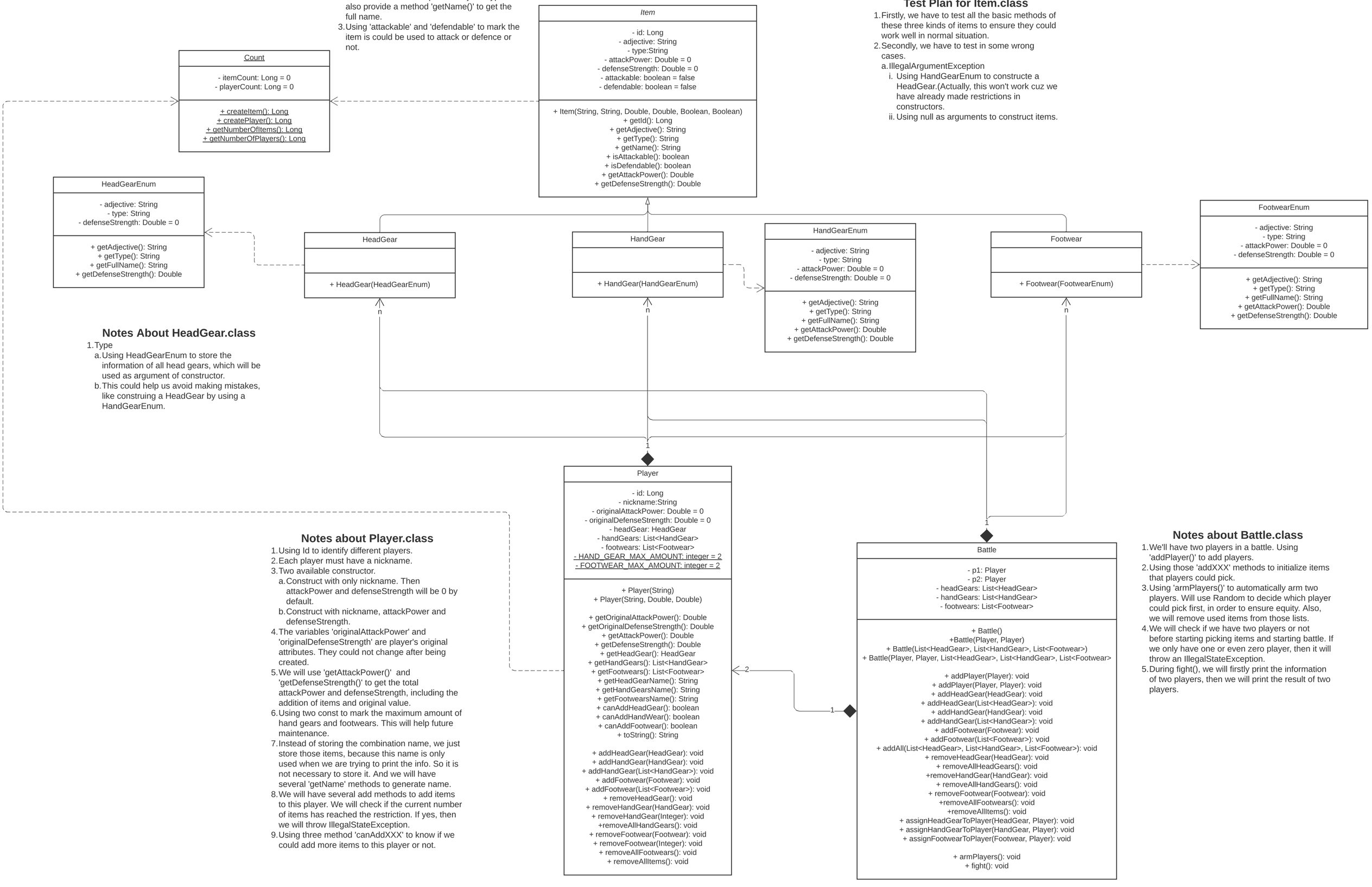
1. Used to count items and players we have created.
2. itemCount and playerCount cannot be modified by other class. They will only increase automatically.
3. Using createItem() to increase itemCount by 1, and will return the newest id. Same as createPlayer().
4. Using getNumberOfItems() to get the number of items we have created. Same as getNumberofPlayers().

Notes About Item.class

1. Using Id to identify different item.
2. Instead of storing the full name of an item, we store the name in two parts -- adj and type. We also provide a method 'getName()' to get the full name.
3. Using 'attackable' and 'defendable' to mark the item is could be used to attack or defence or not.

Test Plan for Item.class

1. Firstly, we have to test all the basic methods of these three kinds of items to ensure they could work well in normal situation.
2. Secondly, we have to test in some wrong cases.
   a. IllegalArgumentException
      i. Using HandGearEnum to constructe a HeadGear.(Actually, this won't work cuz we have already made restrictions in constructors.
      ii. Using null as arguments to construct items.



Notes About HeadGear.class

1. Type
   a. Using HeadGearEnum to store the information of all head gears, which will be used as argument of constructor.
   b. This could help us avoid making mistakes, like construing a HeadGear by using a HandGearEnum.

Notes about Player.class

1. Using Id to identify different players.
2. Each player must have a nickname.
3. Two available constructor.
   a. Construct with only nickname. Then attackPower and defenseStrength will be 0 by default.
   b. Construct with nickname, attackPower and defenseStrength.
4. The variables 'originalAttackPower' and 'originalDefenseStrength' are player's original attributes. They could not change after being created.
5. We will use 'getAttackPower()' and 'getDefenseStrength()' to get the total attackPower and defenseStrength, including the addition of items and original value.
6. Using two const to mark the maximum amount of hand gears and footwears. This will help future maintenance.
7. Instead of storing the combination name, we just store those items, because this name is only used when we are trying to print the info. So it is not necessary to store it. And we will have several 'getName' methods to generate name.
8. We will have several add methods to add items to this player. We will check if the current number of items has reached the restriction. If yes, then we will throw IllegalStateException.
9. Using three method 'canAddXXX' to know if we could add more items to this player or not.

Test Plan for Player.class

1. Firstly, testing all basic methods.
2. Secondly, testing edge cases.
   a. IllegalArgumentException
      i. call those methods by using null as arguments.
      ii. try to remove an item that does not exist.
   b. IllegalStateException
      i. try to add more than 1 head gear.
      ii. try to add more than 2 hand gears.
      iii. try to add more than 2 footwears.

Notes about Battle.class

1. We'll have two players in a battle. Using 'addPlayer()' to add players.
2. Using those 'addXXX' methods to initialize items that players could pick.
3. Using 'armPlayers()' to automatically arm two players. Will use Random to decide which player could pick first, in order to ensure equity. Also, we will remove used items from those lists.
4. We will check if we have two players or not before starting picking items and starting battle. If we only have one or even zero player, then it will throw an IllegalStateException.
5. During fight(), we will firstly print the information of two players, then we will print the result of two players.

Test Plan for Battle.class

1. Firstly, testing all correct usages.
2. Secondly, testing edge cases.
   a. IllegalArgumentException
      i. call those add methods by using null as arguments.
      ii. add a player twice.(means p1 == p2)
      iii. add an item to list twice.(Cuz each item could only be used once)
      iv. trying to add an item that doesn't exist in those lists.
   b. IllegalStateException
      i. add more than two players
      ii. fight while having less than two players.