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(). **Notes About Item.class** 4. Using getNumberOfItems() to get the number 1. Using Id to identify different item. of items we have created. Same as 2. Instead of storing the full name of an item, we getNumberofPlayers(). store the name in two parts -- adj and type. We **Test Plan for Item.class** also provide a method 'getName()' to get the Item 1. Firstly, we have to test all the basic methods of full name. these three kinds of items to ensure they could 3. Using 'attackable' and 'defendable' to mark the - id: Long item is could be used to attack or defence or work well in normal situation. - adjective: String 2. Secondly, we have to test in some wrong - type:String cases. Count - attackPower: Double = 0 a.IllegalArgumentException - defenseStrength: Double = 0 i. Using HandGearEnum to constructe a - itemCount: Long = 0 - attackable: boolean = false HeadGear.(Actually, this won't work cuz we - playerCount: Long = 0 - defendable: boolean = false have already made restrictions in constructors. + createItem(): Long + Item(String, String, Double, Double, Boolean, Boolean) ii. Using null as arguments to construct items. + createPlayer(): Long + getId(): Long + getAdjective(): String + getNumberOfItems(): Long + getNumberOfPlayers(): Long + getType(): String + getName(): String + isAttackable(): boolean + isDefendable(): boolean + getAttackPower(): Double + getDefenseStrength(): Double HeadGearEnum - adjective: String FootwearEnum - type: String - defenseStrength: Double = 0 - adjective: String HandGearEnum - type: String HandGear Footwear HeadGear - attackPower: Double = 0 + getAdjective(): String adjective: String - defenseStrength: Double = 0 + getType(): String - type: String + getFullName(): String - attackPower: Double = 0 + getDefenseStrength(): Double - defenseStrength: Double = 0 + getAdjective(): String + HandGear(HandGearEnum) + Footwear(FootwearEnum) + HeadGear(HeadGearEnum) + getType(): String + getFullName(): String + getAdjective(): String + getAttackPower(): Double + getType(): String + getDefenseStrength(): Double + getFullName(): String + getAttackPower(): Double **Notes About HeadGear.class** + getDefenseStrength(): Double 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. Player - id: Long - nickname:String - originalAttackPower: Double = 0 - originalDefenseStrength: Double = 0 - headGear: HeadGear Notes about Battle.class **Notes about Player.class** - handGears: List<HandGear> - footwears: List<Footwear> 1. We'll have two players in a battle. Using 1. Using Id to identify different players. Battle - HAND_GEAR_MAX_AMOUNT: integer = 2 'addPlayer()' to add players. 2. Each player must have a nickname. - FOOTWEAR MAX AMOUNT: integer = 2 2. Using those 'addXXX' methods to initialize items - p1: Player 3. Two available constructor. that players could pick. p2: Player a. Construct with only nickname. Then 3. Using 'armPlayers()' to automatically arm two - headGears: List<HeadGear> + Player(String) attackPower and defenseStrength will be 0 by - handGears: List<HandGear> players. Will use Random to decide which player + Player(String, Double, Double) default - footwears: List<Footwear> could pick first, in order to ensure equity. Also, b. Construct with nickname, attackPower and + getOriginalAttackPower(): Double we will remove used items from those lists. defenseStrength. + getOriginalDefenseStrength(): Double 4. We will check if we have two players or not 4. The variables 'original Attack Power' and + Battle() + getAttackPower(): Double before starting picking items and starting battle. If +Battle(Player, Player) 'originalDefenseStrength' are player's original + getDefenseStrength(): Double + Battle(List<HeadGear>, List<HandGear>, List<Footwear>) we only have one or even zero player, then it will attributes. They could not change after being + getHeadGear(): HeadGear + Battle(Player, Player, List<HeadGear>, List<HandGear>, List<Footwear> throw an IllegalStateException. created. + getHandGears(): List<HandGear> 5. During fight(), we will firstly print the information 5. We will use 'getAttackPower()' and + getFootwears(): List<Footwear> + addPlayer(Player): void of two players, then we will print the result of two + getHeadGearName(): String getDefenseStrength()' to get the total + addPlayer(Player Player), void + getHandGearsName(): String players. attackPower and defenseStrength, including the + addHeadGear(HeadGear): void + getFootwearsName(): String addition of items and original value. + addHeadGear(List<HeadGear>): void + canAddHeadGear(): boolean + addHandGear(HandGear): void 6. Using two const to mark the maximum amount of + canAddHandWear(): boolean + addHandGear(List<HandGear>): void hand gears and footwears. This will help future + canAddFootwear(): boolean + addFootwear(Footwear): void maintenance. + toString(): String + addFootwear(List<Footwear>): void 7. Instead of storing the combination name, we just + addAll(List<HeadGear>, List<HandGear>, List<Footwear>): void store those items, because this name is only + addHeadGear(HeadGear): void + removeHeadGear(HeadGear): void + addHandGear(HandGear): void used when we are trying to print the info. So it is + removeAllHeadGears(): void + addHandGear(List<HandGear>): void not necessary to store it. And we will have +removeHandGear(HandGear): void + addFootwear(Footwear): void several 'getName' methods to generate name. + removeAllHandGears(): void + addFootwear(List<Footwear>): void + removeFootwear(Footwear): void 8. We will have several add methods to add items + removeHeadGear(): void +removeAllFootwears(): void to this player. We will check if the current number + removeHandGear(HandGear): void +removeAllItems(): void of items has reached the restriction. If yes, then + removeHandGear(Integer): void + assignHeadGearToPlayer(HeadGear, Player): void we will throw IllegalStateException. +removeAllHandGears(): void + assignHandGearToPlayer(HandGear, Player): void 9. Using three method 'canAddXXX' to know if we + removeFootwear(Footwear): void + assignFootwearToPlayer(Footwear, Player): void + removeFootwear(Integer): void could add more items to this player or not. + removeAllFootwears(): void + armPlayers(): void + removeAllItems(): void + fight(): void **Test Plan for Player.class** 1. Firstly, testing all basic methods. **Test Plan for Battle.class** 2. Secondly, testing edge cases. 1. Firstly, testing all correct usages. a. Illegal Argument Exception 2. Secondly, testing edge cases. i. call those methods by using null as arguments. a. IllegalArgumentException ii. try to remove an item that does not exist. i. call those add methods by using null as b.IllegalStateException arguments. i. try to add more than 1 head gear. ii. add a player twice.(means p1 == p2) ii. try to add more than 2 hand gears. iii. add an item to list twice.(Cuz each item could only iii. try to add more than 2 footwears. be used once) iv. trying to add an item that doesn't exist in those

b.IllegalStateException

i. add more than two players

ii. fight while having less than two players.

Notes About Count.class

1. Used to count items and players we have