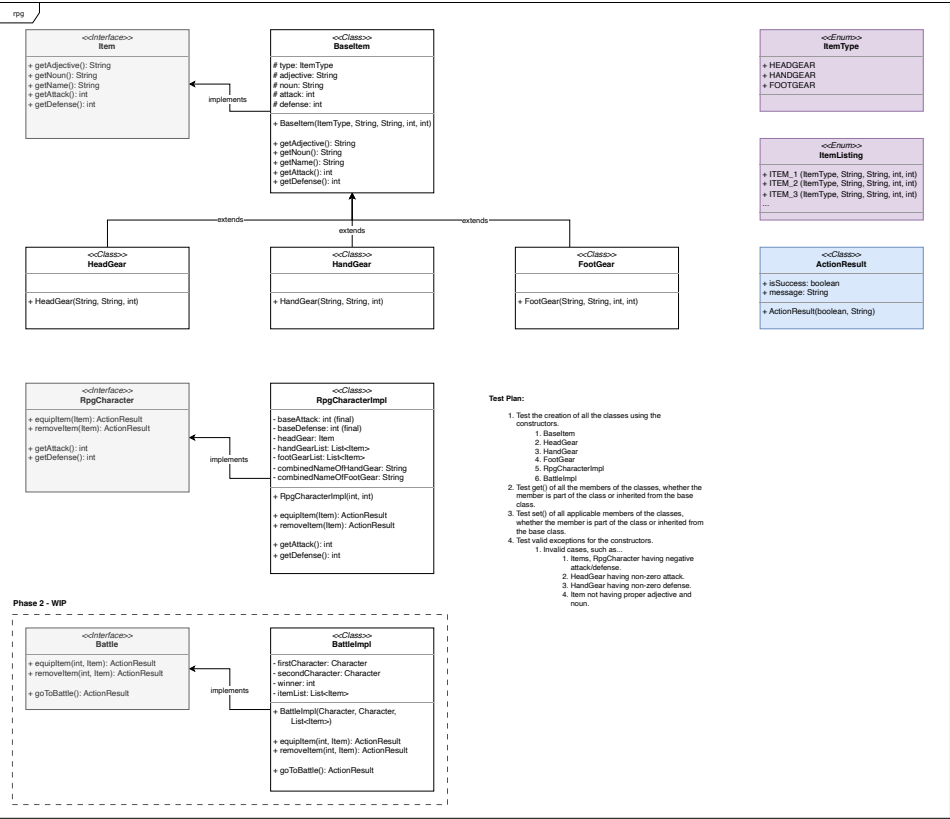# PDP - Lab 2 - UML Design

Version 1
(Naive Implementation)

Notes: Please look at Version 2 for the latest iteration. The initial iteration is shown to showcase the trail of thought.

rpg

<<Interface>>
Item

+ getAdjective(): String
+ getNoun(): String
+ getName(): String
+ getAttack(): int
+ getDefense(): int

implements

<<Class>>
BaseItem

type: ItemType
adjective: String
noun: String
attack: int
defense: int

+ BaseItem(ItemType, String, String, int, int)

+ getAdjective(): String
+ getNoun(): String
+ getName(): String
+ getAttack(): int
+ getDefense(): int

<<Enum>>
ItemType

+ HEADGEAR
+ HANDGEAR
+ FOOTGEAR

<<Enum>>
ItemListing

+ ITEM_1 (ItemType, String, String, int, int)
+ ITEM_2 (ItemType, String, String, int, int)
+ ITEM_3 (ItemType, String, String, int, int)
...

<<Class>>
ActionResult

+ isSuccess: boolean
+ message: String

+ ActionResult(boolean, String)

extends ... *extends* ... *extends*

<<Class>>
HeadGear

+ HeadGear(String, String, int)

<<Class>>
HandGear

+ HandGear(String, String, int)

<<Class>>
FootGear

+ FootGear(String, String, int, int)

<<Interface>>
RpgCharacter

+ equipItem(Item): ActionResult
+ removeItem(Item): ActionResult

+ getAttack(): int
+ getDefense(): int

implements

<<Class>>
RpgCharacterImpl

- baseAttack: int (final)
- baseDefense: int (final)
- headGear: Item
- handGearList: List<Item>
- footGearList: List<Item>
- combinedNameOfHandGear: String
- combinedNameOfFootGear: String

+ RpgCharacterImpl(int, int)

+ equipItem(Item): ActionResult
+ removeItem(Item): ActionResult

+ getAttack(): int
+ getDefense(): int

Test Plan:

1. Test the creation of all the classes using the constructors.
   1. BaseItem
   2. HeadGear
   3. HandGear
   4. FootGear
   5. RpgCharacterImpl
   6. BattleImpl
2. Test get() of all the members of the classes, whether the member is part of the class or inherited from the base class.
3. Test set() of all applicable members of the classes, whether the member is part of the class or inherited from the base class.
4. Test valid exceptions for the constructors.
   1. Invalid cases, such as...
      1. Items, RpgCharacter having negative attack/defense.
      2. HeadGear having non-zero attack.
      3. HandGear having non-zero defense.
      4. Item not having proper adjective and noun.

Phase 2 - WIP

<<Interface>>
Battle

+ equipItem(int, Item): ActionResult
+ removeItem(int, Item): ActionResult

+ goToBattle(): ActionResult

implements

<<Class>>
BattleImpl

- firstCharacter: Character
- secondCharacter: Character
- winner: int
- itemList: List<Item>

+ BattleImpl(Character, Character, List<Item>)

+ equipItem(int, Item): ActionResult
+ removeItem(int, Item): ActionResult

+ goToBattle(): ActionResult

Version 2
(Implementation with Design Patterns)

Notes: Design Patterns adopted: Singleton, Builder, Factory, Composite, Decorator (in principle while doing Composite), Strategy (inbuilt as there is only 1 'naming' strategy).

rpg

<<Interface>>
Item

+ getAdjective(): String
+ getNoun(): String
+ getName(): String
+ getAttack(): int
+ getDefense(): int

implements

<<Class>>
BaseItem

type: ItemType
adjective: String
noun: String
attack: int
defense: int

+ BaseItem(ItemType, String, String, int, int)

+ getAdjective(): String
+ getNoun(): String
+ getName(): String
+ getAttack(): int
+ getDefense(): int

implements

<<Class>>
CombinedItem

- type: ItemType
- combinedAdjectives: String
- finalNoun: String
- combinedAttack: int
- combinedDefense: int
- combinedItemList: List<Item>

+ CombinedItem(ItemType, ItemType)

+ getAdjective(): String
+ getNoun(): String
+ getName(): String
+ getAttack(): int
+ getDefense(): int

<<Enum>>
ItemType

+ HEADGEAR
+ HANDGEAR
+ FOOTGEAR

<<Enum>>
ItemListing

+ ITEM_1 (ItemType, String, String, int, int)
+ ITEM_2 (ItemType, String, String, int, int)
+ ITEM_3 (ItemType, String, String, int, int)
...

+ createItem(): Item

<<Class>>
ActionResult

+ isSuccess: boolean
+ message: String

+ ActionResult(boolean, String)

contains

<<Class (Singleton)>>
ItemListing

- instance: ItemInventory
- itemList: List<Item>

- ItemInventory()

+ getInstance(): ItemInventory
+ getItemList(): List<Item>
+ addItem(Item)

creates

extends ... *extends* ... *extends*

<<Class>>
HeadGear

+ HeadGear(String, String, int)

<<Class>>
HandGear

+ HandGear(String, String, int)

<<Class>>
FootGear

+ FootGear(String, String, int, int)

<<Interface>>
RpgCharacter

+ equipItem(Item): ActionResult
+ removeItem(Item): ActionResult

+ getAttack(): int
+ getDefense(): int

implements

<<Class>>
RpgCharacterImpl

- baseAttack: int (final)
- baseDefense: int (final)
- headGear: Item
- handGearList: List<Item>
- footGearList: List<Item>

+ RpgCharacterImpl(int, int)

+ equipItem(Item): ActionResult
+ removeItem(Item): ActionResult

+ getAttack(): int
+ getDefense(): int

creates

<<Class>>
RpgCharacterBuilder

- baseAttack: int (final)
- baseDefense: int (final)

+ build(int, int): RpgCharacterImpl
+ setBaseAttack(int)
+ setBaseDefense(int)

Test Plan:

1. Test the creation of all the classes using the constructors.
   1. BaseItem
   2. HeadGear
   3. HandGear
   4. FootGear
   5. CombinedItem
   6. RpgCharacterImpl
   7. BattleImpl
2. Test get() of all the members of the classes, whether the member is part of the class or inherited from the base class.
   1. Verify attributes like attack, defense, adjective, noun, and combined attributes (for CombinedItem).
3. Test set() of all applicable members of the classes, whether the member is part of the class or inherited from the base class.
   1. Validate updates to mutable fields in classes like RpgCharacterImpl and ensure correctness.
4. Test valid exceptions for the constructors.
   1. Invalid cases, such as...
      1. Items, RpgCharacter having negative attack/defense.
      2. HeadGear having non-zero attack.
      3. HandGear having non-zero defense.
      4. Item not having proper adjective and noun.
5. Test Builder pattern.
   1. Ensure that the RpgCharacterImpl is not built with invalid (negative, null, etc.) baseAttack and baseDefense.
6. Test Factory pattern.
   1. Ensure that the Items are correctly created based on the ItemListing ENUM.
7. Test Singleton pattern.
   1. Ensure that the same instance is returned, so that the inventory is maintained.
8. Test Composite pattern.
   1. Ensure that the 2 item objects combined are of the same ItemType. Else, throw an exception.
   2. Ensure that the CombinedItem is created properly based on the combination of 2 Item objects.
      1. BaseItem + BaseItem
      2. CombinedItem + BaseItem
   3. Ensure that the overall list of Item(s) is maintained.
   4. Ensure that the attributes are 'passed' on properly, based on the 'naming' strategy.
9. Ensure that the RpgCharacter has max 1 HeadGear, 2 HandGear(s), 2 FootGear(s).

Phase 2 - WIP

<<Interface>>
Battle

+ equipItem(int, Item): ActionResult
+ removeItem(int, Item): ActionResult

+ goToBattle(): ActionResult

implements

<<Class>>
BattleImpl

- firstCharacter: Character
- secondCharacter: Character
- winner: int
- itemList: List<Item>

+ BattleImpl(Character, Character, List<Item>)

+ equipItem(int, Item): ActionResult
+ removeItem(int, Item): ActionResult

+ goToBattle(): ActionResult