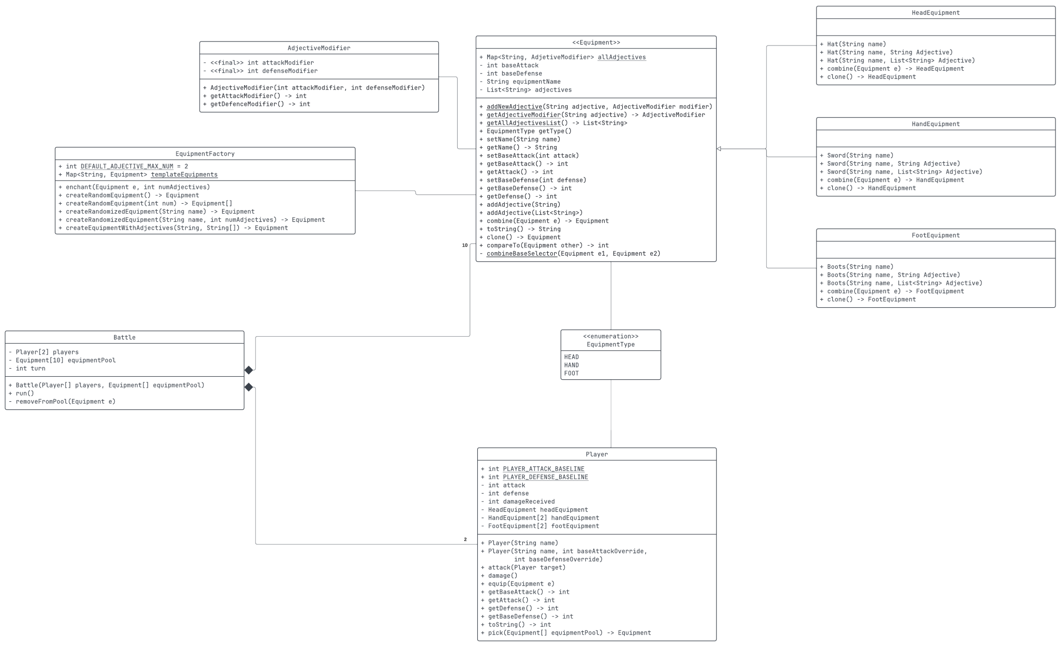
Lab02. Role Playing Games

UML


AdjectiveModifier
- <<final>> int attackModifier
- <<final>> int defenseModifier
+ AdjectiveModifier(int attackModifier, int defenseModifier)
+ getAttackModifier() -> int
+ getDefenceModifier() -> int
EquipmentFactory
+ int DEFAULT_ADJECTIVE_MAX_NUM = 2
+ Map<String, Equipment> templateEquipments
+ enchant(Equipment e, int numAdjectives)
+ createRandomEquipment() -> Equipment
+ createRandomEquipment(int num) -> Equipment[]
+ createRandomizedEquipment(String name) -> Equipment
+ createRandomizedEquipment(String name, int numAdjectives) -> Equipment
+ createEquipmentWithAdjectives(String, String[]) -> Equipment
<<Equipment>>
+ Map<String, AdjetiveModifier> allAdjectives
- int baseAttack
- int baseDefense
- String equipmentName
- List<String> adjectives
+ addNewAdjective(String adjective, AdjectiveModifier modifier)
+ getAdjectiveModifier(String adjective) -> AdjectiveModifier
+ getAllAdjectivesList() -> List<String>
+ EquipmentType getType()
+ setName(String name)
+ getName() -> String
+ setBaseAttack(int attack)
+ getBaseAttack() -> int
+ getAttack() -> int
+ setBaseDefense(int defense)
+ getBaseDefense() -> int
+ getDefense() -> int
+ addAdjective(String)
+ addAdjective(List<String>)
+ combine(Equipment e) -> Equipment
+ toString() -> String
+ clone() -> Equipment
+ compareTo(Equipment other) -> int
- combineBaseSelector(Equipment e1, Equipment e2)
HeadEquipment
+ Hat(String name)
+ Hat(String name, String Adjective)
+ Hat(String name, List<String> Adjective)
+ combine(Equipment e) -> HeadEquipment
+ clone() -> HeadEquipment
HandEquipment
+ Sword(String name)
+ Sword(String name, String Adjective)
+ Sword(String name, List<String> Adjective)
+ combine(Equipment e) -> HandEquipment
+ clone() -> HandEquipment
FootEquipment
+ Boots(String name)
+ Boots(String name, String Adjective)
+ Boots(String name, List<String> Adjective)
+ combine(Equipment e) -> FootEquipment
+ clone() -> FootEquipment
Battle
- Player[2] players
- Equipment[10] equipmentPool
- int turn
+ Battle(Player[] players, Equipment[] equipmentPool)
+ run()
- removeFromPool(Equipment e)
10
<<enumeration>>
EquipmentType
HEAD
HAND
FOOT
Player
+ int PLAYER_ATTACK_BASELINE
+ int PLAYER_DEFENSE_BASELINE
- int attack
- int defense
- int damageReceived
- HeadEquipment headEquipment
- HandEquipment[2] handEquipment
- FootEquipment[2] footEquipment
+ Player(String name)
+ Player(String name, int baseAttackOverride, int baseDefenseOverride)
+ attack(Player target)
+ damage()
+ equip(Equipment e)
+ getBaseAttack() -> int
+ getAttack() -> int
+ getDefense() -> int
+ getBaseDefense() -> int
+ toString() -> int
+ pick(Equipment[] equipmentPool) -> Equipment
2

Approach and Testing Plans

Class `Equipment`

- The class maintains a HashMap to map adjectives to a pair of modifiers representing a adjective's effect on the stats of an equipment
- The class only maintains a *base* value of equipment stats. The values after modifier is calculated dynamically upon the call to `getAttack()` and `getDefense()`
- There can be unlimited number of *different* adjective to an equipment. Their effects stack.
- Two same-type equipment can `combine()`, which combines their base stats value and adjectives

Testing Plan

- Getter and Setter methods

- Construct each non-abstract class with no modifiers

  Expected: Default values are applied correctly for each class of equipment
- Construct a known-good class of equipment, then test how adjective modifiers affect the stats of an equipment

  Expected: The adjective applies intended modified value to the result of `getAttack()` and `getDefense()`

Combining Equipments

The requirement did not specify which item currently equipping will get combined, quote

> To be clever, when a character picks up two items of the same type, their names are combined, they make a new piece of footwear that combines the powers and name. (Tom's note: let's make the assumption that combining only happens when the player doesn't have empty slot for that gear) The new name is the adjective from one item and the full name from the other.

Since the combination only happens where there are already 2 items for `HandEquipment` and `FootEquipment`, I assume always the last item is getting combined.
I also designed a FIFO behavior to the item slots of these type to make sure all equipment will have chance to get combined.

- Test two equipment of same-type equipment combine (different adjectives)

  Expected:
  A new equipment with two adjective and a combined stats;
  Item name change matches the requirement.
- Test the combination of two pieces of equipment with the same adjective.

  Expected:
  The new piece of equipment should only contain one instance of the adjective, and the effect of the adjective should only be applied once to its

stats.
Item name change matches the requirement.

- Test the combination of two pieces of equipment where one has no adjectives.

  Expected:
  No adjectives on combined equipment, but the stats is stacked.
  Item name change matches the requirement.

- Test the combination of more than two pieces of equipment.

  Expected:
  The new piece of equipment should contain all unique adjectives from the combined equipment, and the effects of all adjectives should be applied to its stats.
  Item name change matches the requirement.

Class `Player`

- The class represents a player who has a base attack, defense and HP.
- By default all players are created using the same set of parameters, but this can be overridden.
- `Player` have 1 slot for `headEquipment`, 2 slots each for `HandEquipment` and `FootEquipment`

Testing Plan

- Getter and Setter methods
- Constructor with no parameters

  Expected: A player which stats match the global default.

- Constructor with parameters

  Expected: Corresponding stats gets overridden

- `ObtainEquipment()`
  - If the obtained equipment can be equipped to an empty slot:

    Expected:
    An empty slot of the type of the equipment is filled with the equipment;
    The player stats is adjusted accordingly.

- If the obtained equipment cannot be equipped to an empty slot:

  Expected:
  An equipment currently holding the last slot of that type is picked and combined with the new equipment;
  The other item is moved to the last slot of that type;
  Newly combined item will occupy the first slot of that type;

  The player stats is adjusted accordingly.