Homework 2

Role Play Design Document

Student:

Lillith Chute

Teacher:

Clark Freifeld

Course:

CS 5010

Design Summary

1. Because the nature of the application breaks down the combat simulator into three aspects (armor, adventurer, and combat) it made sense conceptually to create interfaces for those particular domains.

2. Armor seems to be the most complex as that general concept is broken up into three types of armor. Specifically, head, hands, and feet. Armor has basic stats, but each *type* of armor is a little different. Therefore, make armor abstract and then implement the differences in concrete classes for the different types.

3. The adventurer and the combat doesn't have the same level of complexity so I don't see why abstraction would be necessary in those instances.

Test Plan

Do the basics:

1. Make sure the tests are not in the same package as the code.
2. Test that the constructors for the various classes work to included error checking and validations.
3. Testing getter and setter methods are generally straightforward.
4. Check code coverage to make sure as many lines of code have been checked as possible.

5. As a note, I test private methods by making them public if they are complex and then setting them back to private. There is probably a better way, but that's the quick and dirty way for me at the moment.

Difficulties

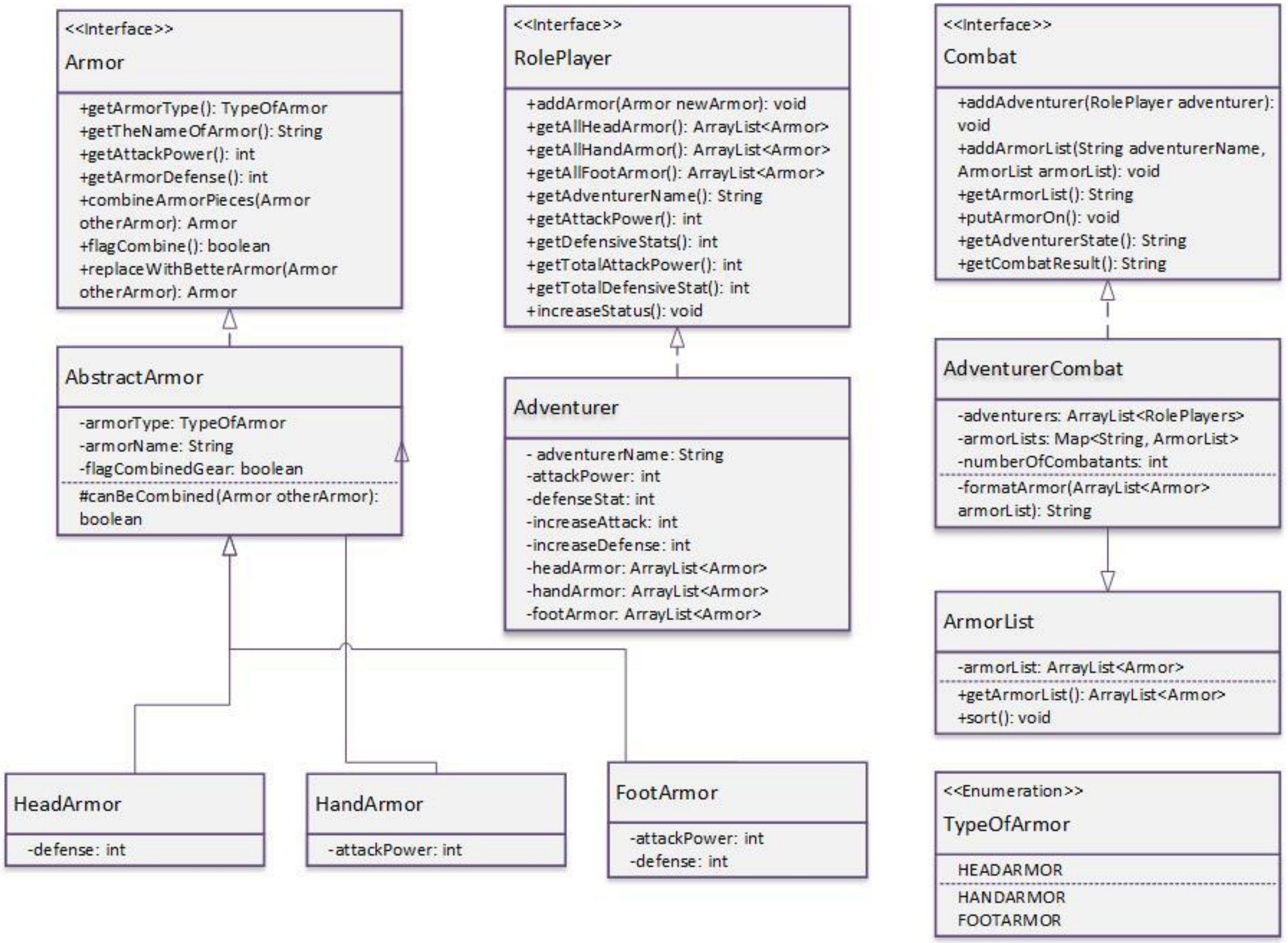
Some of the public methods are pretty complicated and these are the ones I did my best to pay attention to. They are as follows:

1. *combineArmorPieces(Armor otherArmor)*
2. getCombatResult()
3. putArmorOn()
4. replaceWithBetterArmor(Armor otherArmor)

Notes

Due to time constraints, I was not able to test absolutely everything. It is too complicated an application to test every possible scenario with all of the variability inherent in the system. Furthermore, it would have been nice to set up a more interesting driver program, but again, limitations of time and practicality did not allow for more flexible and interesting interactive program. Had there been more time both of those items would have been much richer.

UML Diagram


<<Interface>>
Armor
+getArmorType(): TypeOfArmor
+getTheNameOfArmor(): String
+getAttackPower(): int
+getArmorDefense(): int
+combineArmorPieces(Armor otherArmor): Armor
+flagCombine(): boolean
+replaceWithBetterArmor(Armor otherArmor): Armor
<<Interface>>
RolePlayer
+addArmor(Armor newArmor): void
+getAllHeadArmor(): ArrayList<Armor>
+getAllHandArmor(): ArrayList<Armor>
+getAllFootArmor(): ArrayList<Armor>
+getAdventurerName(): String
+getAttackPower(): int
+getDefensiveStats(): int
+getTotalAttackPower(): int
+getTotalDefensiveStat(): int
+increaseStatus(): void
<<Interface>>
Combat
+addAdventurer(RolePlayer adventurer): void
+addArmorList(String adventurerName, ArmorList armorList): void
+getArmorList(): String
+putArmorOn(): void
+getAdventurerState(): String
+getCombatResult(): String
AbstractArmor
-armorType: TypeOfArmor
-armorName: String
-flagCombinedGear: boolean
#canBeCombined(Armor otherArmor): boolean
Adventurer
- adventurerName: String
-attackPower: int
-defenseStat: int
-increaseAttack: int
-increaseDefense: int
-headArmor: ArrayList<Armor>
-handArmor: ArrayList<Armor>
-footArmor: ArrayList<Armor>
AdventurerCombat
-adventurers: ArrayList<RolePlayers>
-armorLists: Map<String, ArmorList>
-numberOfCombatants: int
-formatArmor(ArrayList<Armor> armorList): String
ArmorList
-armorList: ArrayList<Armor>
+getArmorList(): ArrayList<Armor>
+sort(): void
HeadArmor
-defense: int
HandArmor
-attackPower: int
FootArmor
-attackPower: int
-defense: int
<<Enumeration>>
TypeOfArmor
HEADARMOR
HANDARMOR
FOOTARMOR