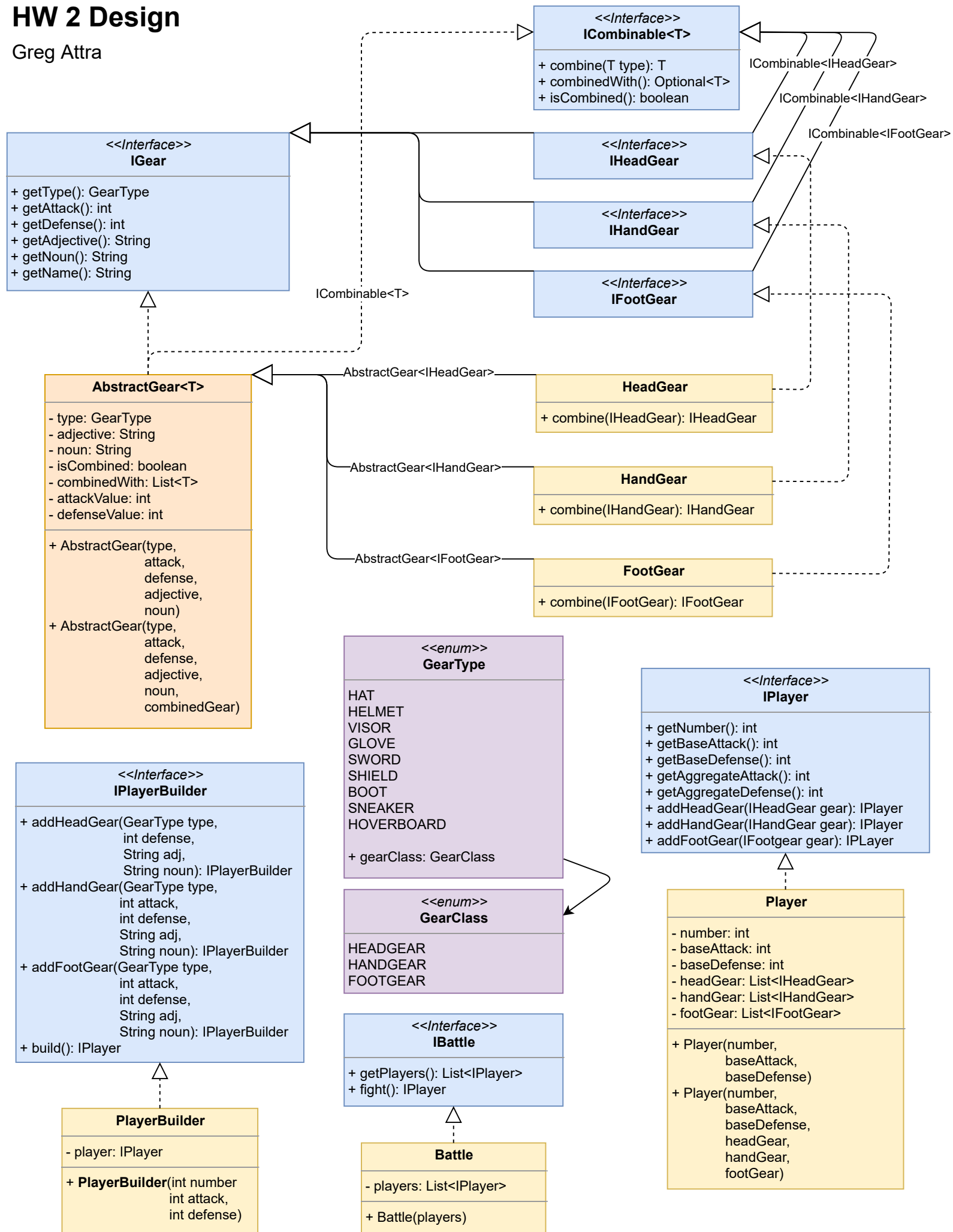


HW 2 Design

Greg Attra



Homework 2 Testing Strategy

Greg Attra

09/28/2020

Below is the testing strategy for each class in the implementation of homework 2.

AbstractGear<T>

AbstractGear class to hold common functionality between different `IGear` implementations. `T` is a generic placeholder for the `IGear` subinterface implemented by extending class.

Constructor:

- AbstractGear(type, attack, defense, adjective, noun)
 - `type` property is of type GearType (enum)
 - `attack` is an `int` that must be non-negative
 - `defense` is an `int` that must be non-negative
 - `adjective` is a single-word `String` that must not be empty
 - `noun` is a single-word `String` that must not be empty
- Gear(type, attack, defense, adjective, noun, combinedWith)
 - `type` property is of type GearType (enum)
 - `attack` is an `int` that must be non-negative
 - `defense` is an `int` that must be non-negative
 - `adjective` is a single-word `String` that must not be empty
 - `noun` is a single-word `String` that must not be empty
 - `combinedWith` is an instance of `T` interface which must be set if using this constructor

getType(): GearType

- Accessor for GearType property
- Assert returned type matches that of constructor arg

getAttack(): int

- Returns the aggregate `attack` `int` value (player's base `attack` + sum of gear `attack` values)
- Assert that value returned matches expected given base `attack` and added `gears`

getDefense(): int

- Returns the aggregate `attack` `int` value (player's base `attack` + sum of gear `attack` values)
- Assert that value returned matches expected given base `attack` and added `gears`

getAdjective(): String

- Returns the `adjective` `String` passed into the constructor
- Assert returned value matches value passed into constructor

getNoun(): String

- Returns the `noun` `String` value passed into constructor
- Assert returned value matches value passed into constructor

getName(): String

- Returns the combination of `getAdjective` and `getNoun`
- Assert correct values when:

- Not a combined gear
- A combined gear is set

combinedWith(): Optional<List<T>>

- Returns the `combinedWith` property, if set, otherwise null
- Assert returned `List<T>` equals actual `combinedWith` gear using overridden `toString()` and `equals()` and `hashCode()` methods

isCombined(): boolean

- Returns `true` if `combinedWith` is set to an `IGear` instance
- Returns `false` if `combinedWith` is `null`

combine(T): T

- Factory method instantiating a new `T` instance with `combinedWith` set to the passed-in `T` instance
- To be implemented by extending classes.

HeadGear, HandGear, FootGear

These classes implement `AbstractGear<T>` passing in their respective interfaces as `T`. The testing strategy for these classes will include all the scenarios listed for the `AbstractGear<T>` abstract class. Plus unique functionality to each concrete class.

combine(T): T (where T is either IHeadGear, IHandGear, IFootGear)

- Factory method instantiating a new `T` instance with `combinedWith` set to the passed-in `T` instance
- If `T` instance is not of the same `GearType` as the argument `T` instance, throw `IllegalArgumentException`
- If `isCombined()` already `true`, this gear has already been combined. Throw `IllegalStateException`
 - Test for when called instance is already combined
 - Test for when passed-in instance is already combined

equals(Object other)

- Override which compares `other`'s `toString()` method with called instance's `toString()` method

Player

Player class for the RPG. Can be dressed with various gears which augment its defense and attack power.

Constructor:

- `Player(number, baseAttack, baseDefense)`
 - `number` is non-negative `int`
 - `baseAttack` is non-negative `int`
 - `baseDefense` is non-negative `int`
- `Player(number, baseAttack, baseDefense, headGear, handGear, footGear)`
 - `number` is non-negative `int`
 - `baseAttack` is non-negative `int`
 - `baseDefense` is non-negative `int`

- `headGear` is a `IGear` with length exactly 1 (no more, no less)
 - Assert `gear` instance is of type `HeadGear`
- `handGear` is a `List<IGear>` with length between 1 and 2
 - Assert `gear` instances are of type `HandGear`
- `footGear` is a `List<IGear>` with length between 1 and 2
 - Assert `gear` instances are of type `FootGear`

getNumber(): int

- Returns the player's `number` which was set by constructor
- Assert returned value matches value passed into constructor

getAttack(): int

- Aggregates the sum of the `attack` value for each `gear` set on the Player, plus the Player's base `attack` value set by constructor
- Assert return value matches expected using hard-coded sums in test cases

getDefense(): int

- Aggregates the sum of the `defense` value for each `gear` set on the Player, plus the Player's base `defense` value set by constructor
- Assert return value matches expected using hard-coded sums in test cases

addHeadGear(IHeadGear gear): IPlayer

- Factory method which adds a headgear to the player and returns the new player instance.
- Test that add succeeds when:
 - Player has no headgear set
 - Current headgear has not been combined and the gear being added is of the same GearType
- Test that add fails when:
 - Player has a headgear set and that headgear is of a different type
 - Player has a headgear set and that headgear is already combined

addHandGear(IHandGear gear): IPlayer

- Factory method which adds a handgear to the player and returns the new player instance.
- Test that add succeeds when:
 - Player has no handgear set
 - Current handgear has not been combined and the gear being added is of the same GearType
- Test that add fails when:
 - Player has a handgear set and that handgear is of a different type
 - Player has a handgear set and that handgear is already combined

addFootGear(IFootGear gear): IPlayer

- Factory method which adds a footgear to the player and returns the new player instance.
- Test that add succeeds when:
 - Player has no footgear set
 - Current footgear has not been combined and the gear being added is of the same GearType
- Test that add fails when:

- Player has a footgear set and that footgear is of a different type
- Player has a footgear set and that footgear is already combined

toString(): String

- Represents the Player object in String format
- Assert that returned String matched expected (use hard-coded String in test case)

equals(Object other): boolean

- Returns `true` if `other` matches instance being called
- Used `hashCode()` of the `toString()` response `String`
- Test returns `true` for Player objects with same states
- Test returns `false` for Player objects with different states

Battle

Class for orchestrating battles between players. Sets a specified number of players and pits them against one another until only one is victorious.

constructor():

- Takes the following params:
 - A non-negative `int` for `playerCount` which dictates how many players must participate in the fight
 - A list of `IPlayer` instances. Can be empty.

getPlayers()

- Getter method for the players set on this instance
- Assert that the values returned matched values passed to constructor

addPlayer(IPlayer player): IBattle

- Factory method which adds a player to the list of players and returns a new updated Battle instance
- Test that method runs successfully when:
 - The player limit has not been met
- Test that method fails when:
 - The player limit has been met
 - A player with the same number has already been added

fight()

- Pits each player in the list against one another until a victor is determined
- Assert that the most powerful player wins in a list of two players
- Assert that the most powerful player wins in a list of three players

PlayerBuilder

Builder class for Player instance.

constructor():

- Takes the following params:
 - Non-negative `int` number
 - Non-negative `int` attack
 - Non-negative `int` defense

addHeadGear(GearType type, int defense, String adj, String noun): IPlayerBuilder

- Instantiates an IHeadGear instance using the provided args and attempts to add to the `player` instance it is building
- Test method runs successfully when:
 - Args are valid and Player has space or an uncombined gear
- Test method fails when:
 - GearType invalid
 - Negative `attack` provided
 - Negative `defense` provided
 - Empty `adj` value
 - Empty `noun` value
 - Player has no room or has only combined gear

addHandGear(GearType type, int defense, String adj, String noun): IPlayerBuilder

- Instantiates an IHandGear instance using the provided args and attempts to add to the `player` instance it is building
- Test method runs successfully when:
 - Args are valid and Player has space or an uncombined gear
- Test method fails when:
 - GearType invalid
 - Negative `attack` provided
 - Negative `defense` provided
 - Empty `adj` value
 - Empty `noun` value
 - Player has no room or has only combined gear

addFootGear(GearType type, int defense, String adj, String noun): IPlayerBuilder

- Instantiates an IFootGear instance using the provided args and attempts to add to the `player` instance it is building
- Test method runs successfully when:
 - Args are valid and Player has space or an uncombined gear
- Test method fails when:
 - GearType invalid
 - Negative `attack` provided
 - Negative `defense` provided
 - Empty `adj` value
 - Empty `noun` value
 - Player has no room or has only combined gear