Homework 2 Testing Strategy
Greg Attra
09/28/2020

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

Gear
Constructor:
- Gear(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 `IGear` interface which must be set if using this constructor

getBaseAttack(): int
- Returns the `attack` `int` value passed into constructor
- Assert that value returned matches value passed into constructor

getBaseDefense(): int
- Returns the `defense` `int` value passed into constructor
- Assert that value returned matches value passed into constructor

getAggregateAttack(): int
- Returns the `attack` value passed into constructor, plus the `attack` value of the gear with which it is combined, if one exists
- In practice it will add the combined `gear`'s `getAggregateAttack()` return value to its base `attack` value
- Assert correct values when:
    - No combined gear set
    - A combined gear is set

getAggregateDefense(): int
- Returns the `defense` value passed into constructor, plus the `defense` value of the gear with which it is combined, if one exists
- In practice it will add the combined `gear`'s `getAggregateDefense()` return value to its base `defense` value
- Assert correct values when:
    - No combined gear set
    - A combined gear is set

getBaseAdjective(): String

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

getAggregateAdjective(): String

- Returns a comma-separated `String` of the `gear`'s `adjective` value, plus the `adjective` of the gear with which it is combined, if one exists
- In practice it will add the combined `gear`'s `getAggregateAdjective()` return value to is base `adjective` value
- Assert correct values when:
    - No combined gear set
    - A combined gear is set

getNoun(): String

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

getName(): String

- Returns the combination of `getAggregateAdjective` and `getName`
- Assert correct values when:
    - No combined gear set
    - A combined gear is set

combinedWith(): Optional<IGear>

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

isCombined(): boolean

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

combine(IGear): IGear

- Factory method instantiating a new `IGear` instance with `combinedWith` set to the passed-in `IGear` instance
- If `IGear` instance is not of the same type as the argument `IGear` 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

Player

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`