- **Design Explanation:**

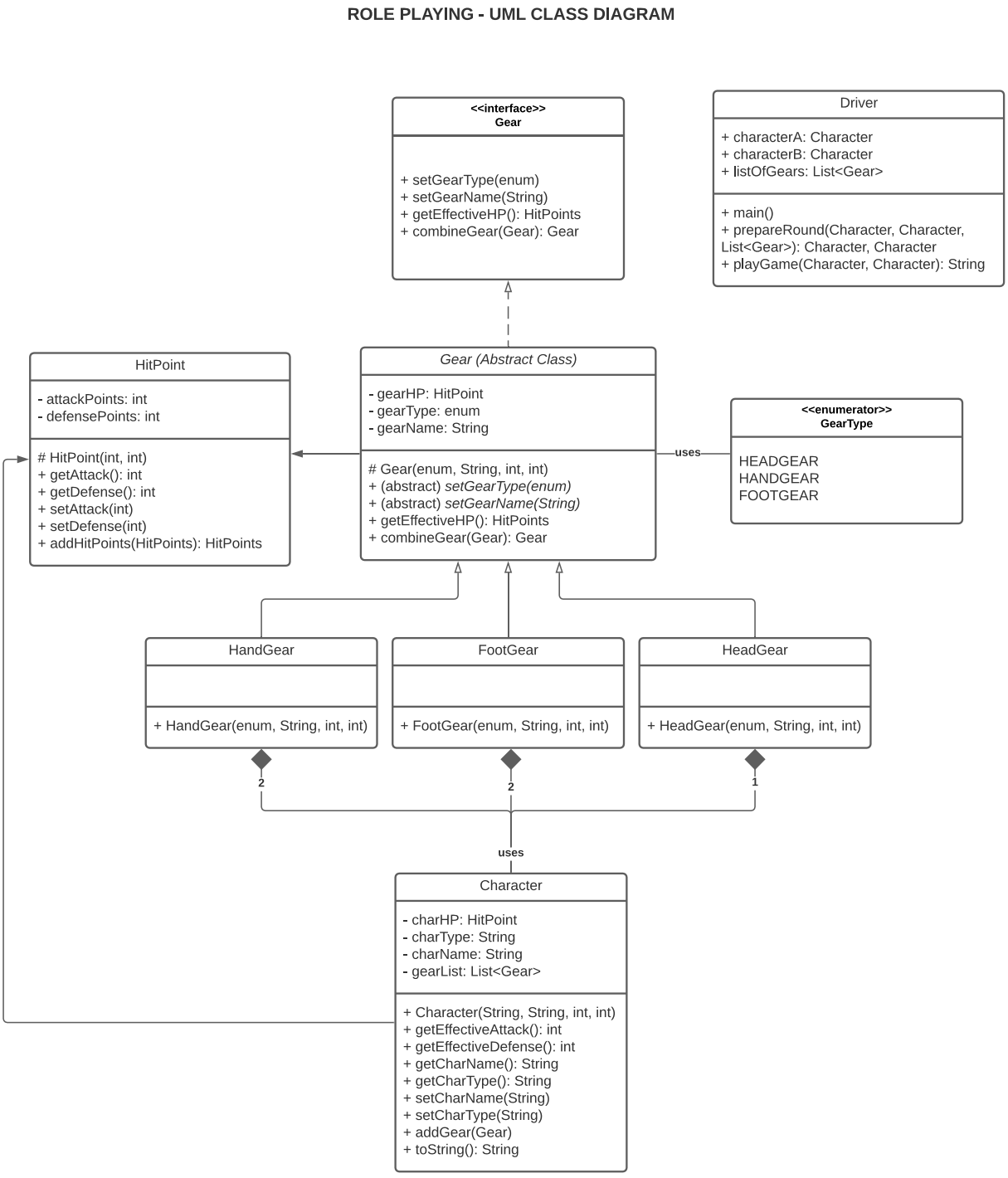
  - **Package Name:** roleplaying
  - **Character Class:**
    - The character class consists of fields (Like Name of the character, type of character, attack points and defense points) that describe a character of the game.
    - The character uses HitPoint object to define its attack and defense points.
    - The character has the capacity to pick/choose gears from a list of 10 gears available that can be equipped to increase the hitpoint stats of the character. This can be done using the **addGear(Gear)** method of this class. The constraints are it can't pick more than 2 hand gears, 2 foot gears and 1 head gear.
    - It has the ability to combine 2 gears of same type. The method **combineGear(Gear)** of the gear object is used to perform the same.
    - **getEffectiveAttack() and getEffectiveDefense()** methods will give the user the increased attack points and increased defense points respectively after the character has equipped the gears.
    - The **setter methods** are used to set values (define) of each fields of a character.
    - The **getter methods** are used to view what values are set for the fields of a character.
    - The chacter class has '***has-a***' relationship with the gear class via the gear interface.

  - **Gear Interface:**
    - The gear interface consists of all the methods that needs to be implemented by the gear class.
    - The gear class is defined as abstract to offer encapsulation and enable DRY (Don't Repeat Yourself) for code refactoring.
    - The gear class uses HitPoint object to define the attack and defense points of the gears.
    - The gear class has children (sub-class) belonging to each class of gears namely: Head Gear. Hand Gear, Foot Gear.
    - The children class inherits all the fields and methods of the parent class.
    - The parent abstract gear class has 2 **abstract setter methods** that needs to be implemented in the children class. They are the type of gear and name of the gear. The type of gear is choosen from the **GearType** enumerator.
    - The **getEffectiveHP()** method is used to get the HP of the gear (applicable to both raw gear and combined super gears).
    - **combineGear(Gear)** is used to combine to gears of same type. Throws an exception if 2 different types of gear are provided. If valid, creates a new combined gear with new name and HP.

  - **Driver Class:**
    - The driver class is where the main function resides.
    - The main function is where a quick game demo is played. That is, objects of class gears and characters are instantiated and are used to play the game.
    - The main function uses 2 helper methods. prepareRound(Character, Character, List<Gear>) and playGame(Character, Character).
    - **prepareRound(Character, Character, List<Gear>)** is used to prepare the players for battle. That is this is where the players choose gears to upgrade themselves.

- **playGame(Character, Character)** is where the game between the 2 upgraded characters is played. Here based on their hit points a player is given the title of winner. The outcome of the duel is displayed here.

- ## UML Class Diagram:



- **Testing Plan:**
  - **Test 1:** To check if an object of class character is created properly.
  - **Test 2:** To check if an object of class HandGear is created properly.
  - **Test 3:** To check if an object of class HeadGear is created properly.
  - **Test 4:** To check if an object of class FootGear is created properly.
  - **Test 5:** To check if an object of class HitPoints is created properly.
  - **Test 6:** To check if an object of class HitPoints can be added to another object of the same class.
  - **Test 7:** To check if an exception is thrown if an object of class HitPoints is added to object of type Integer or some other data type.
  - **Test 8:** To check if 2 gears of same type are combined properly.
  - **Test 9:** To check if an exception is thrown if 2 gears of different types are combined.
  - **Test 10:** To check if an exception is thrown if the character tries to pick more than 2 hand gears at a time.
  - **Test 11:** To check if an exception is thrown if the character tries to pick more than 2 foot gears at a time.
  - **Test 12:** To check if an exception is thrown if the character tries to pick more than 1 head gear at a time.
  - **Test 13:** To check if a character's hit-points have changed accordingly after the character has been equipped with some gears.
  - **Test 14:** To check if toString method of the Character class is providing the expected output.
  - **Test 15:** To check if player 1 has won.
  - **Test 16:** To check if player 2 has won.
  - **Test 17:** To check if the game has been tied.

- **Challenges Faced:**
  - Various assumptions that makes the above choices dependent of them.
    - Whether same items will be available for the second character who picks the items at last.
    - Whether a character can take up to 8 items {3 hand gears (1 combo, 1 raw), 3 foot gears (1 combo, 1 raw), 2 head gears (1 combo)}.
  - Use of relationships between classes were a challenge. That is, whether to use "Association" or "Composition" relationship between the character and the gear.
  - To define the access modifiers(visibility) of the class methods.