

Problem Description [Due: 3 May]

Big fan of Stranger Things on Netflix? Then, you should be pretty familiar with “Dungeons and Dragons”, which is basically a pen and paper fantasy role-playing game. Prior to playing a game of D&D each player needs to create their own character, which embark upon imaginary adventures within a fantasy setting. In this assignment, you are given the task of designing these characters

Solution Description

You are given an abstract class `Character.java` and an interface `Race.java`. Implement all the missing methods of the abstract class, and the subclasses `Fighter.java`, `Rogue.java`, `Wizard.java`, and `Cleric.java`. You will need to create the subclasses from scratch.

Character.java

- A character has several instance variables relating to the characters abilities. They are:
 1. name
 2. level
 3. 6 ability scores (strength, dexterity, constitution, intelligence, wisdom and charisma)
 4. 6 ability score modifiers (strengthMod, dexterityMod, constitutionMod, intelligenceMod, wisdomMod and charismaMod)
 5. 6 ability score increases (strengthInc, dexterityInc, constitutionInc, intelligenceInc, wisdomInc and charismaInc)
 6. proficiencyMod
 7. health
 8. race
 9. isDead
- Implement the getter and setter methods that have been partially provided.
- Two constructors
 1. Assigning `String name`, `int level`, `int strength`, `int dexterity`, `int constitution`, `int intelligence`, `int wisdom`, `int charisma`, `int proficiencyMod`, `int health`, `boolean isDead`. The health variable should start equal to five times the character level, and should never go above that. If the health variable ever goes below 0, isDead should be set to true and the health variable be reset to 0. The race variable should be one of the instance variables of interface `Race`. The `proficiencyMod` variable should be set to +2. For instance variables such as `strengthInc`, `dexterityInc`, `constitutionInc`, `intelligenceInc`, `wisdomInc`, `charismaInc`, check Table3. Finally, for instance variables such as `strengthInc`, `dexterityInc`, `constitutionInc`, `intelligenceInc`, `wisdomInc`, `charismaInc`, consider Table1.

Table1

Ability	Modifier	Ability	Modifier
2 – 3	-4	12 – 13	1
4 – 5	-3	14 – 15	2
6 – 7	-2	16 – 17	3

8 – 9	-1	18 – 19	4
10 – 11	0	20 – 21	5

2. Only taking in a name and a seed, sets the level to one, sets health to 5, and randomly sets the abilities as following:
 - a. Roll a dice four times (End result should be 4 numbers)
 - b. Drop the lowest number
 - c. Add the remaining numbers together
 - d. Repeat five more times, for a total of six numbers and set the abilities.

You can perform 2(a) by randomly generating a number between 1 and 6 (inclusive) by using a Random object using the given seed. Based on your character (Fighter or Cleric) you will put your two highest scores to two specific abilities and assign the rest 4 numbers to the rest of the abilities in this order [[strength](#), [dexterity](#), [constitution](#), [intelligence](#), [wisdom](#) and [charisma](#)], e.g. let's say you have generated 12, 11, 14, 7, 9 4 and you are instructed to assign the two largest score to ability [strength](#) and [wisdom](#). Hence [strength](#) and [wisdom](#) will be set to 14, 12 and the others [dexterity](#), [constitution](#), [intelligence](#) and [charisma](#) will be set to 11, 7, 9 and 4 respectively. For information on which two abilities should be assigned highest score check Table2.

Based on your race, your final ability (all or specific ones) will increase according to Table2. E.g, if your calculated [strength](#) is 10 and [strengthInc](#) is 3 based on your race, the final strength should be 13.

Fighter.java

- This class should extend [Character](#) and implement [Race](#).
- Implement 2 constructors corresponding with each of the constructors of the superclass.
- Implement the [attack\(Character c\)](#) method. The [attack](#) method of Fighter should decrement the [health](#) of the parameter [character](#) by 10 plus the fighter's [strength](#) variable. If the character's [isDead](#) variable is true, this method should do nothing besides printing "Cannot attack a dead character" to the console.
- Implement the [levelUp\(\)](#) method. This method should increase the character's [level](#) by 1, reset the [health](#) to it's maximum (5 times the level), increase [strength](#) by 2, and all other abilities by 1.
- a [toString\(\)](#) method that returns a string with format "Level (level) fighter named (name) with (strength) strength, (dexterity) dexterity, (constitution) constitution, (intelligence) intelligence, (wisdom) wisdom and (charisma) charisma"

Rogue.java

- This class should extend [Character](#) and implement [Race](#).
- Implement 2 constructors corresponding with each of the constructors of the superclass.
- Implement the [attack\(Character c\)](#) method. The [attack](#) method of [Rogue](#) should decrement the [health](#) of the parameter character by 6 plus the rogue's [dexterity](#) variable. If the

character's `isDead` variable is true, this method should do nothing besides printing "Cannot attack a dead character" to the console.

- Implement the `levelUp()` method. This method should increase the character's `level` by 1, reset the `health` to it's maximum (5 times the level), increase `dexterity` by 3, and all other abilities by 2.
- a `toString()` method that returns a string with format "Level (level) fighter named (name) with (strength) strength, (dexterity) dexterity, (constitution) constitution, (intelligence) intelligence, (wisdom) wisdom and (charisma) charisma"

Cleric.java

- This class should extend `Character` and implement `Race`.
- Implement 2 constructors corresponding with each of the constructors of the superclass.
- Implement the `attack(Character c)` method. The `attack` method of `Cleric` should decrement the `health` of the parameter character by 6 plus the cleric's `wisdom` variable. If the character's `isDead` variable is true, this method should do nothing besides printing "Cannot attack a dead character" to the console.
- Implement the `levelUp()` method. This method should increase the character's `level` by 1, reset the `health` to it's maximum (5 times the `level`), increase `wisdom` by 2, and all other abilities by 1.
- Create a new method called `heal(Character c)` which increases the parameter characters `health` by 6 plus the cleric's `wisdom` variable, but not beyond their maximum `health` (5 times their level). If a character is dead, do nothing except print "Cannot heal a dead character" to the console.
- a `toString()` method that returns a string with format "Level (level) fighter named (name) with (strength) strength, (dexterity) dexterity, (constitution) constitution, (intelligence) intelligence, (wisdom) wisdom and (charisma) charisma"

Wizard.java

- This class should extend `Character` and implement `Race`.
- Implement 2 constructors corresponding with each of the constructors of the superclass.
- Implement the `attack(Character c)` method. The `attack` method of `Wizard` should decrement the `health` of the parameter character by a 4 plus the wizard's `intelligence` variable. If the character's `isDead` variable is true, this method should do nothing besides printing "Cannot attack a dead character" to the console.
- Implement the `levelUp()` method. This method should increase the character's `level` by 1, reset the `health` to it's maximum (5 times the level), increase `intelligence` by 2, and all other stats by 1.
- Create a new method called `multiAttack(Character[] c)` which decreases each character in the parameter's `health` by 2 plus the wizard's `intelligence` variable (different values for each character). If a character is dead, do nothing to that character except print "Cannot damage a dead character" to the console.
- a `toString()` method that returns a string with format "Level (level) fighter named (name) with (strength) strength, (dexterity) dexterity, (constitution) constitution, (intelligence) intelligence, (wisdom) wisdom and (charisma) charisma"

Table2

Character	Ability with the highest score	Ability with the 2 nd highest score
Fighter	strength	dexterity
Rogue	dexterity	intelligence
Cleric	wisdom	strength or constitution
Wizard	intelligence	constitution or dexterity

Table3

Race	Ability increase (increase methods)
Human	All (+1)
Halfling	Dexterity (+2)
Elf	Dexterity (+2)
Dwarf	Constitution (+2)

Running and Testing

Design your own Game class with a main method to create a simulation to test of all the methods for bonus: 5pts. However, while checking your assignment, I will run my own Game class to check your assignments.

Grading

Will be announced later.

Instruction

- Create a file named your 10 digit student Id.java (i.e., 2021-3-60-000.java) and copy classes [Fighter](#), [Rogue](#), [Wizard](#), and [Cleric](#) into that file. You can also add Game class into that file if you have implemented it as a bonus.
- Any sort of plagiarism will be dealt very stricly with negative marking