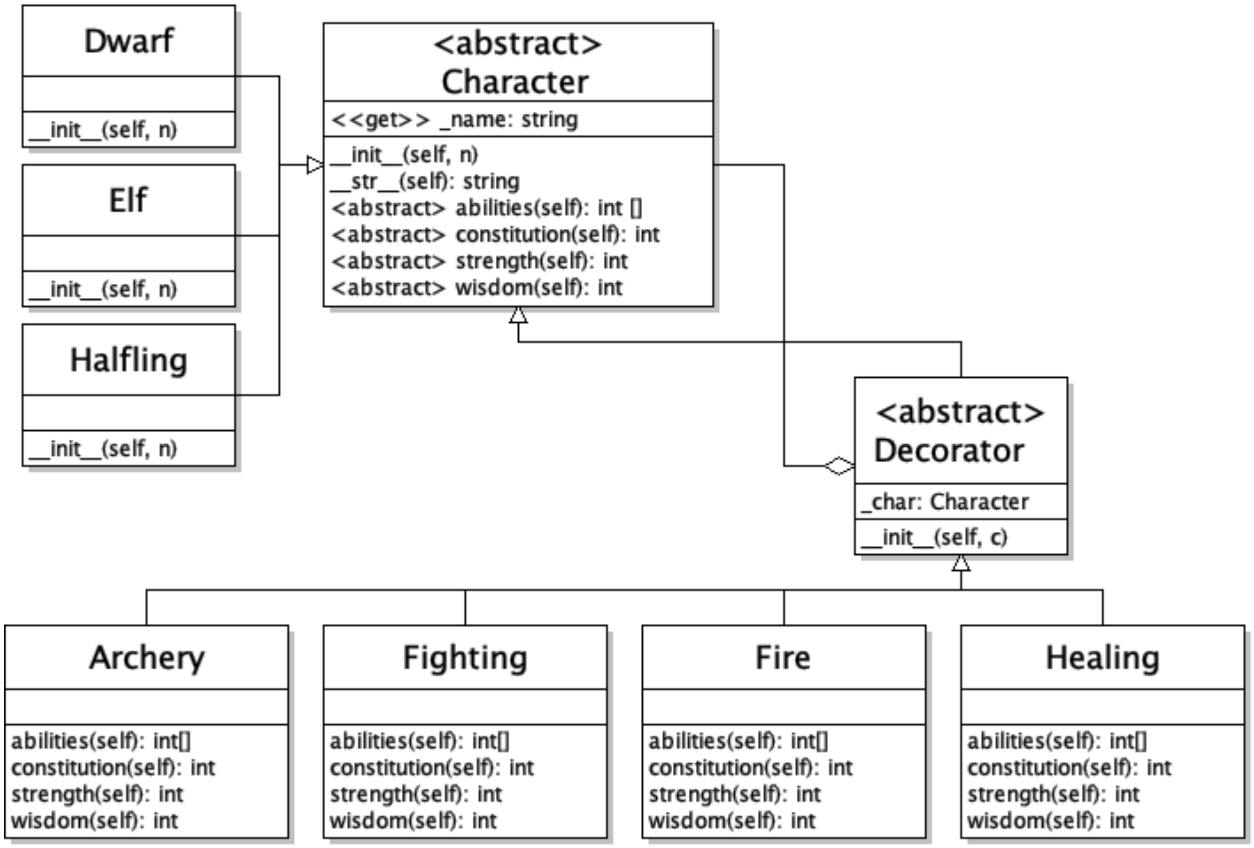
CECS 277 – Lab 11 – Decorator Pattern

Character Maker

Using the Decorator pattern, create a role playing character creation program that uses three base types of characters (Dwarf, Elf, Halfling). Those characters can then be decorated with four different abilities (Archery, Fighting, Fire, Healing). The program will then display the updated stats for the created character.

Use the following UML and class descriptions to create your program:



Classes:

1. Character - rather than an interface, the Character class should be abstract since the character has a name attribute.
   a. Use the property decorators to make a getter for the name attribute.
   b. __init__(self, n) – set the name attribute using the parameter n.
   c. __str__(self) – return a string with the name, abilities, and stats.
   d. abilities(self), constitution(self), strength(self), and wisdom(self) – abstract methods (no code)
2. Concrete Characters (Dwarf/Elf/Halfling) – extend Character.
   a. __init__(self, n) – uses super to initialize the name with the character type.
   b. abilities(self) – returns the starting list of levels of the four different abilities.
   c. constitution(self), strength(self), and wisdom(self) – returns the starting stats.

3. Decorator – abstract and extends from Character.
    a. __init__(self, c) – since Character is an abstract class rather than an interface, you need to call super init so the decoration has the name attribute. Pass it the name of the character parameter, then set the character attribute to c.
    b. abilities(self), constitution(self), strength(self), and wisdom(self) – call the same method on your character attribute.
4. Ability Decoration Classes (Archery/Fighting/Fire/Healing) – extend Decorator
    a. abilities(self) – add 1 to the ability level for this ability.
    b. constitution(self), strength(self), and wisdom(self) – call super for each method and add/subtract the additional stat value for this ability.
5. Main – present the user with a menu to choose the base character type and prompt them for their character's name. Construct a character of that type and then display it. Then, prompt the user to add 5 abilities to the character. For each one, decorate the character with that ability and display the updated stats. When they are finished, check if any of the stats are <= 0, if they are, then they have failed at making a character, otherwise, they were successful.

Example Output:

```
Character Maker!
Choose your character:
1. Dwarf
2. Elf
3. Halfling
Enter choice: 2

What is your character's name? Larue

Larue the Elf
-Abilities-
Archery:  Level 1
-Stats-
Con: 13
Str: 10
Wis: 15

You have 5 points to spend:
Add an ability:
1. Archery
2. Fighting
3. Fire Magic
4. Healing
Enter ability: 1
Larue the Elf
-Abilities-
Archery:  Level 2
-Stats-
Con: 11
Str: 15
Wis: 13

Add an ability:
1. Archery
2. Fighting
3. Fire Magic
4. Healing
Enter ability: 2
Larue the Elf
-Abilities-
Archery:  Level 2
Fighting: Level 1
-Stats-
Con: 13
Str: 17
Wis: 10

...

Add an ability:
1. Archery
2. Fighting
3. Fire Magic
4. Healing
Enter ability: 4
Larue the Elf
-Abilities-
Archery:  Level 2
Fighting: Level 1
Fire Magic: Level 1
Healing: Level 2
-Stats-
Con: 16
Str: 8
Wis: 19

Congratulations!  You have created
your character.
```

Notes:

1. You should have 10 different files: main.py, character.py, dwarf.py, elf.py, halfling.py, decorator.py, archery.py, fighting.py, fire.py, and healing.py.
2. Check all user input using the get_int_range function in the check_input module.
3. Please do not create any extra methods, attributes, functions, parameters, etc.
4. Please do not create any global variables or use any of the attributes globally (ie. do not access any of the attributes using the underscores, use the properties instead).
5. Use docstrings to document each of the classes, their attributes, and their methods.
6. Place your names, date, and a brief description of the program in a comment block at the top of your main file. Place brief comments throughout your code.
7. The attributes list is the number of levels the character has in each of the abilities [archery, fighting, fire, healing]. For example, if the attribute list is [0,1,2,0], then the character is a Level 1 Fighter, and a Level 2 Fire Mage. If you like, you may start your characters with an ability.
8. You can decide what values to use for your stats and abilities, but try to make sure that the values are balanced (ie. if one stat goes up, another should go down). If needed, here are some suggested starting values for your characters:

	attributes	constitution	strength	wisdom
Dwarf	[0,1,0,0]	13	15	10
Elf	[1,0,0,0]	13	10	15
Halfling	[0,0,0,1]	15	10	13

   and some suggested values for each additional level of ability:

	attributes	constitution	strength	wisdom
Archery	[+1,0,0,0]	-2	+5	-2
Fighting	[0,+1,0,0]	+2	+2	-3
Fire	[0,0,+1,0]	-3	-1	+5
Healing	[0,0,0,+1]	+3	-4	+2

9. Thoroughly test your program before submitting:
   a. Make sure that each character starts with the correct name, abilities, and stats.
   b. Make sure that every time the user decorates the character, the abilities and stats are updated with the correct values.
   c. Make sure that the program ends after the user has chosen 5 abilities.
   d. Make sure to display whether the user successfully created their character or not.

Character Maker – Time estimate: 4 hours

Character Maker **10 points**	Correct. 2 points	A minor mistake. 1.5 points	A few mistakes. 1 point	Several mistakes. 0.5 points	No attempt. 0 points
Character classes (separate files)**:** 1. Character class is abstract with init, str, and the four abstract methods. 2. Dwarf/Elf/Halfling extend from Character. 3. init uses super to set name with type. 4. Correctly overrides methods with starting values. 5. attributes method returns a list.					
Decorator (separate file)**:** 1. Inherits from ABC and Character. 2. init passes in character, calls super to set name, and sets character attribute. 3. Overrides the four methods and calls the method on the attribute.					
Ability Decorations (separate files)**:** 1. Ability classes extend the decorator. 2. Overrides the four methods that each call super and then add/subtract the appropriate values.					
Main file (in separate file)**:** 1. Allows user to choose a character. 2. Allows user to add 5 abilities. 3. Displays the updated stats. 4. Displays whether the character creation was successful (stats all > 0). 5. Error checks all user input.					
Code Formatting: 1. Correct documentation. 2. Meaningful variable names. 3. No exceptions thrown. 4. No global variables or accessing attributes directly. 5. Correct spacing.					