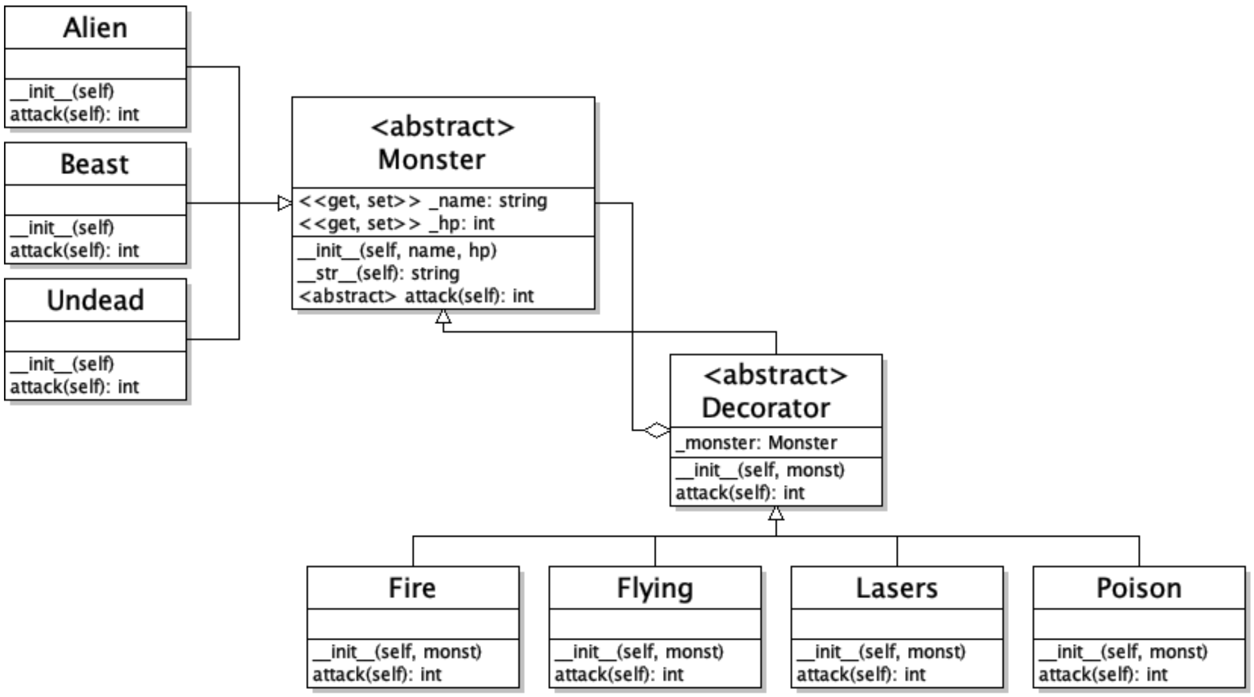
CECS 277 – Lab 13 – Decorator Pattern

Monster Maker

Using the Decorator pattern, create a monster make program that uses three base types of monsters (ie. Alien, Beast, Undead). Those monsters can then be decorated with four different monster abilities (ie. Fire, Lasers, Flying, Poison). The program will then display the updated stats for the constructed monster.

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



Classes:

1. Monster - rather than an interface, the Monster class should be abstract since the monster has attributes.
    a. Use the property decorators to make a getter and setter for the name and hp.
    b. __init__(self, name, hp) – set the name and hp using the parameters
    c. __str__(self) – return a string with the name, hp, and attack power of the monster.
    d. attack(self) – abstract method (no code) – this returns the attack power of the monster.
2. Concrete Monsters (Alien/Beast/Undead) - extends Monster
    a. __init__(self) – uses super to initialize the base name and hp for that monster.
    b. attack(self) – returns the base attack power of that monster.
    c. Note: you get to decide the monster's default name, hp, and attack power.
3. Decorator – abstract and extends from Monster
    a. __init__(self, monst) – since Monster is an abstract class rather than an interface, you need to call super init to construct the Decorator. Pass it the name and hp of the monster parameter, then set the monster attribute to monst.

©2022 Cleary

   b. attack(self): call attack on your monster attribute.
4. Ability Decoration Classes (Flying/Fire/Poison/Lasers) – extends Decorator
   a. __init__(self, monst) – update the monster's name by attaching the ability name and also give the monster bonus hp for that ability. Call super init and pass it the monster.
   b. attack(self) – call super attack and add on additional attack power for this ability.
   c. Note: you get to decide the default values to add to the monster.
5. Main – present the user with a menu to choose the base monster type. Display the base stats of the monster. Then, prompt the user to add a new ability to the monster, decorate the monster with that ability and display the updated monster. Allow the user to add abilities until they choose to quit, then display their final monster.

Example Output:

```
Monster Maker!
Choose a base monster:
1. Alien
2. Beast
3. Undead
Enter choice: 1

Name: Alien
Hp: 5
Attack: 5
Add an ability:
1. Fire
2. Flying
3. Lasers
4. Poison
5. Quit
Enter ability: 3

Name: Alien with Lasers
Hp: 7
Attack: 8
Add an ability:
1. Fire
2. Flying
3. Lasers
4. Poison
5. Quit
Enter ability: 1

Name: Firey Alien with Lasers
Hp: 9
Attack: 11
Add an ability:
1. Fire
2. Flying
3. Lasers
4. Poison
5. Quit
Enter ability: 4

Name: Poison Firey Alien with
Lasers
Hp: 11
Attack: 14
Add an ability:
1. Fire
2. Flying
3. Lasers
4. Poison
5. Quit
Enter ability: 2

Name: Flying Poison Firey Alien
with Lasers
Hp: 13
Attack: 17
Add an ability:
1. Fire
2. Flying
3. Lasers
4. Poison
5. Quit
Enter ability: 1

Name: Firey Flying Poison Firey
Alien with Lasers
Hp: 15
Attack: 20
Add an ability:
1. Fire
2. Flying
3. Lasers
4. Poison
5. Quit
Enter ability: 1

Name: Firey Firey Flying Poison
Firey Alien with Lasers
Hp: 17
Attack: 23
Add an ability:
```

```
1. Fire
2. Flying
3. Lasers
4. Poison
5. Quit
Enter ability: 5
```

```
Your final monster is:
Name: Firey Firey Flying Poison
Firey Alien with Lasers
Hp: 17
Attack: 23
```

Notes:

1. You should have 10 different files: main.py, monster.py, alien.py, beast.py, undead.py, decorator.py, fire.py, flying.py, lasers.py, and poison.py.
2. Check all user input using the get_int_range function in the check_input module.
3. 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).
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. Give each of your monsters different default names, starting hp, and attack power.
8. Each of your ability classes should add on a different ability name to the monster's name, add on a set number of hp, and a set amount of attack power. Use the monster's setters to update their names and hp.
9. Thoroughly test your program before submitting:
    a. Make sure that each base monster starts with the correct name, hp, and attack power.
    b. Make sure that every time the user decorates the monster, its name, hp, and attack power are updated with the correct values.
    c. Make sure that the program ends when the user chooses to quit.

©2022 Cleary