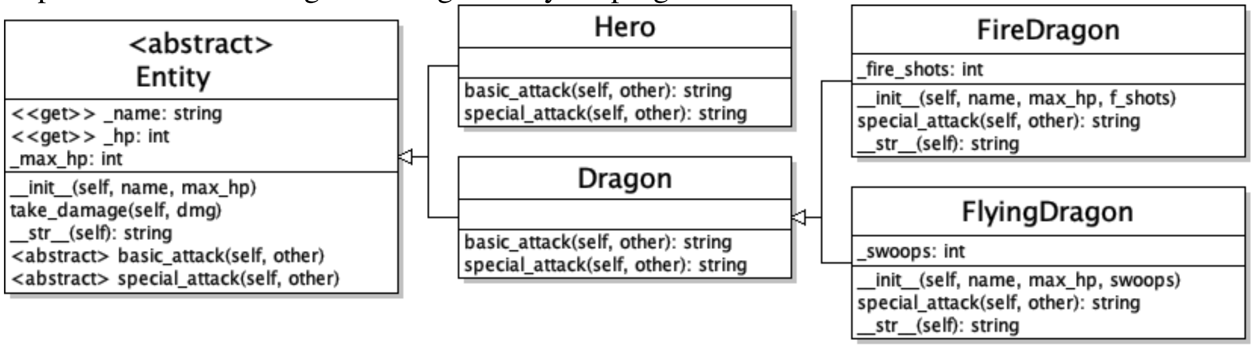
CECS 277 – Lab 8 – Abstract Classes

Dragon Trainer

Create a game where the user must defeat three dragons to pass the trails. Use inheritance to implement the following class diagram in your program.



<abstract>
Entity
<<get>> _name: string
<<get>> _hp: int
_max_hp: int
__init__(self, name, max_hp)
take_damage(self, dmg)
__str__(self): string
<abstract> basic_attack(self, other)
<abstract> special_attack(self, other)
Hero
basic_attack(self, other): string
special_attack(self, other): string
Dragon
basic_attack(self, other): string
special_attack(self, other): string
FireDragon
_fire_shots: int
__init__(self, name, max_hp, f_shots)
special_attack(self, other): string
__str__(self): string
FlyingDragon
_swoops: int
__init__(self, name, max_hp, swoops)
special_attack(self, other): string
__str__(self): string

Entity Class – Abstract class - (in a separate file named: 'entity.py') –
Attributes: _name – entity's name, _hp – entity's hit points, _max_hp – entity's starting hp. Use decorators to create the properties for the name and hp attributes.
Methods:

1. `__init__(self, name, max_hp)` – set the _name, _max_hp, and _hp. Assign the max_hp value to both the _max_hp and _hp attributes.
2. `take_damage(self, dmg)` – the damage the entity takes. Subtract the dmg value from the entity's _hp value. Do not let the hp go past 0 (if it's a negative value, reset it to 0).
3. `__str__(self)` – display the entity's name and hp in the format "Name: hp/max_hp".
4. Abstract methods: `basic_attack(self, other)` and `special_attack(self, other)` – these methods will be overridden by each of the subclasses to create their attacks. Use a decorator to make them abstract.

Hero Class – inherits from Entity – (in a separate file named: 'hero.py') –

1. `basic_attack(self, other)` – the other entity (the dragon) takes a random amount of damage in the range 2D6 (1-6 + 1-6). Return a string with the description of the attack and the damage dealt to the dragon.
2. `special_attack(self, other)` – the other entity (the dragon) takes a random amount of damage in the range 1D12 (1-12). Return a string with the description of the attack and the damage dealt to the dragon.

Dragon Class – inherits from Entity – (in a separate file named: 'dragon.py') –

1. `basic_attack(self, other)` – tail attack – the other entity (the hero) takes a random amount of damage in the range 3-7. Return a string with the description of the attack and the damage dealt to the hero.
2. `special_attack(self, other)` – claw attack – the other entity (the hero) takes a random amount of damage in the range 4-8. Return a string with the description of the attack and the damage dealt to the hero.

FireDragon Class – inherits from Dragon – (in a separate file named: 'fire_dragon.py') –

1. `__init__(self, name, max_hp, f_shots)` – call super init to set the name and hp, then set the number of fire_shots.
2. `special_attack(self, other)`– overridden fire attack – if the dragon has any fire_shots left, then the other entity (the hero) takes a random amount of damage in the

range 5-9 and the number of fire_shots is decremented and a string with the description of the attack and the damage dealt to the hero is returned. Otherwise, no damage is dealt and a string with the description of the failure is returned.

3. `__str__(self)` – use super to get the __str__ from the entity class, then concatenate on the number of fire_shots.

<u>FlyingDragon Class</u> – inherits from Dragon – (in a separate file named: 'flying_dragon.py') –

1. `__init__(self, name, max_hp, swoops)` – call super init to set the name and hp, then set the number of swoops.
2. `special_attack(self, other)` – overridden swoop attack – if the dragon has any swoops left, then the other entity (the hero) takes a random amount of damage in the range 5-8 and the number of swoops is decremented and a string with the description of the attack and the damage dealt to the hero is returned. Otherwise, no damage is dealt and a string with the description of the failure is returned.
3. `__str__(self)` – use super to get the __str__ from the entity class, then concatenate on the number of swoops.

<u>Main</u> (in a separate file named: 'main.py') – Construct a Hero object and then create a list that contains one of each of the different dragons (Dragon, FireDragon, FlyingDragon). Present a menu that allows the user to choose which dragon to attack, and then another menu that gives them the option of attacking with a sword or an arrow (basic or special attack). Call the hero's attack method on the dragon they chose and display the attack message returned. If the user defeats the dragon (hp is 0), then remove that dragon from the list. Then choose a random (surviving) dragon that will attack the user, and randomly choose either a basic or special attack and display the attack message returned. Repeat the attacks until the user defeats all three dragons, or until the hero is knocked out. Check all input for validity.

Example Output (user input is in italics)**:**

```
What is your name, challenger?
Astrid
Welcome to dragon training, Astrid
You must defeat 3 dragons.

Astrid: 50/50
1. Attack Deadly Nadder: 10/10
2. Attack Gronckle: 15/15
Fire Shots remaining: 3
3. Attack Timberjack: 20/20
Swoop attacks remaining: 5
Choose a dragon to attack: 3

Attack with:
1. Sword (2 D6)
2. Arrow (1 D12)
Enter weapon: 2

You slash the Timberjack with your
sword for 9 damage.
Gronckle smashes you with its tail
for 7 damage!

Astrid: 43/50
1. Attack Deadly Nadder: 10/10
2. Attack Gronckle: 15/15
Fire Shots remaining: 3
3. Attack Timberjack: 11/20
Swoop attacks remaining: 5
Choose a dragon to attack: 3

Attack with:
1. Sword (2 D6)
2. Arrow (1 D12)
Enter weapon: 1

You hit the Timberjack with an
arrow for 9 damage.
Gronckle engulfs you in flames for
7 damage!

Astrid: 36/50
1. Attack Deadly Nadder: 10/10
2. Attack Gronckle: 15/15
Fire Shots remaining: 2
3. Attack Timberjack: 2/20
Swoop attacks remaining: 5
Choose a dragon to attack: 3
```

```
Attack with:
1. Sword (2 D6)
2. Arrow (1 D12)
Enter weapon: 2

You slash the Timberjack with your
sword for 5 damage.
Gronckle engulfs you in flames for
5 damage!

Astrid: 31/50
1. Attack Deadly Nadder: 10/10
2. Attack Gronckle: 15/15
Fire Shots remaining: 1
Choose a dragon to attack: 3
Invalid input – should be within
range (1-2).
Choose a dragon to attack: 2

Attack with:
1. Sword (2 D6)
2. Arrow (1 D12)
Enter weapon: 1

You hit the Gronckle with an arrow
for 9 damage.
Deadly Nadder slashes you with its
claws for 8 damage!

Astrid: 23/50
1. Attack Deadly Nadder: 10/10
2. Attack Gronckle: 6/15
Fire Shots remaining: 1
Choose a dragon to attack: 1

Attack with:
1. Sword (2 D6)
2. Arrow (1 D12)
Enter weapon: 2

You slash the Deadly Nadder with
your sword for 11 damage.
Gronckle smashes you with its tail
for 5 damage!

Astrid: 18/50
1. Attack Gronckle: 6/15
Fire Shots remaining: 1
Choose a dragon to attack: 1

Attack with:
1. Sword (2 D6)
2. Arrow (1 D12)
Enter weapon: 2

You slash the Gronckle with your
sword for 9 damage.

Congratulations! You have defeated
all 3 dragons, you have passed the
trials.
```

Notes:

1. You should have 7 different files: entity.py, hero.py, dragon.py, fire_dragon.py, flying_dragon.py, main.py, and check_input.py.
2. Check all user input using the get_int_range function in the check_input module.
3. Do not create any extra attributes, methods, or parameters. You may create additional functions in main.py as needed.
4. Use decorators to make the abstract class, abstract methods, and properties.
5. Please do not create any global variables or use the attributes globally (ie. do not access any of them using the underscore). You can access the name and hp using the get properties, but you shouldn't access the max_hp, fire_shots, or swoops outside of their own classes. You can use __init__ and __str__, when calling the superclass's version.
6. Use docstrings to document each of the classes, their attributes, and each of their methods. See the lecture notes for examples.
7. Place your names, date, and a brief description of your program in a comment block at the top of your program. Place brief comments throughout your code.
8. Thoroughly test your program before submitting:
    a. Make sure that your classes are inherited properly. Hero and Dragon should inherit from Entity, and Fire and Flying Dragons should inherit from Dragon.
    b. Make sure user input is validated. Including when a dragon is defeated and removed from the list, that dragon should no longer be selectable.
    c. Make sure that the damage dealt is correctly subtracted from the opponent.

d. Make sure that the randomly selected dragon is alive.
e. Make sure that the flying/fire dragons cannot do their special attack once they run out of their respective charges (and no damage is dealt).
f. Make sure the game ends when the user defeats all 3 dragons, or when the hero runs out of hp.

Dragon Trainer Rubric – Time estimate: 4 hours

Dragon Trainer **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
Entity class (in a separate file)**:** 1. It is abstract. 2. Has correct attributes and properties. 3. Has correct methods. 4. Has abstract methods for attacks.					
Hero and Dragon classes (sep. files)**:** 1. Inherits from Entity class. 2. Overrides basic_attack method. Does damage to other entity and returns a string describing the attack. 3. Overrides special_attack method. Does damage and returns a string.					
Fire/Flying classes (in separate files)**:** 1. Inherits from Dragon class. 2. Has extra attribute for shots/swoops. 3. Overrides init and calls super. 4. Overrides str method and calls super. 5. Overrides special_attack method, does damage, decrements shots/swoops, and returns attack string.					
Main file (in separate file)**:** 1. Constructs Hero. 2. Constructs all 3 dragons in a list. 3. Has loop to repeat until user dies or all three dragons defeated. 4. Prompts user for dragon choice. 5. Prompts user for attack type. 6. Damage is applied correctly. 7. Removes dragon from list.					
Code Formatting: 1. Correct spacing. 2. Meaningful variable names. 3. No exceptions thrown. 4. No global variables or accessing attributes directly. 5. Correct documentation.					