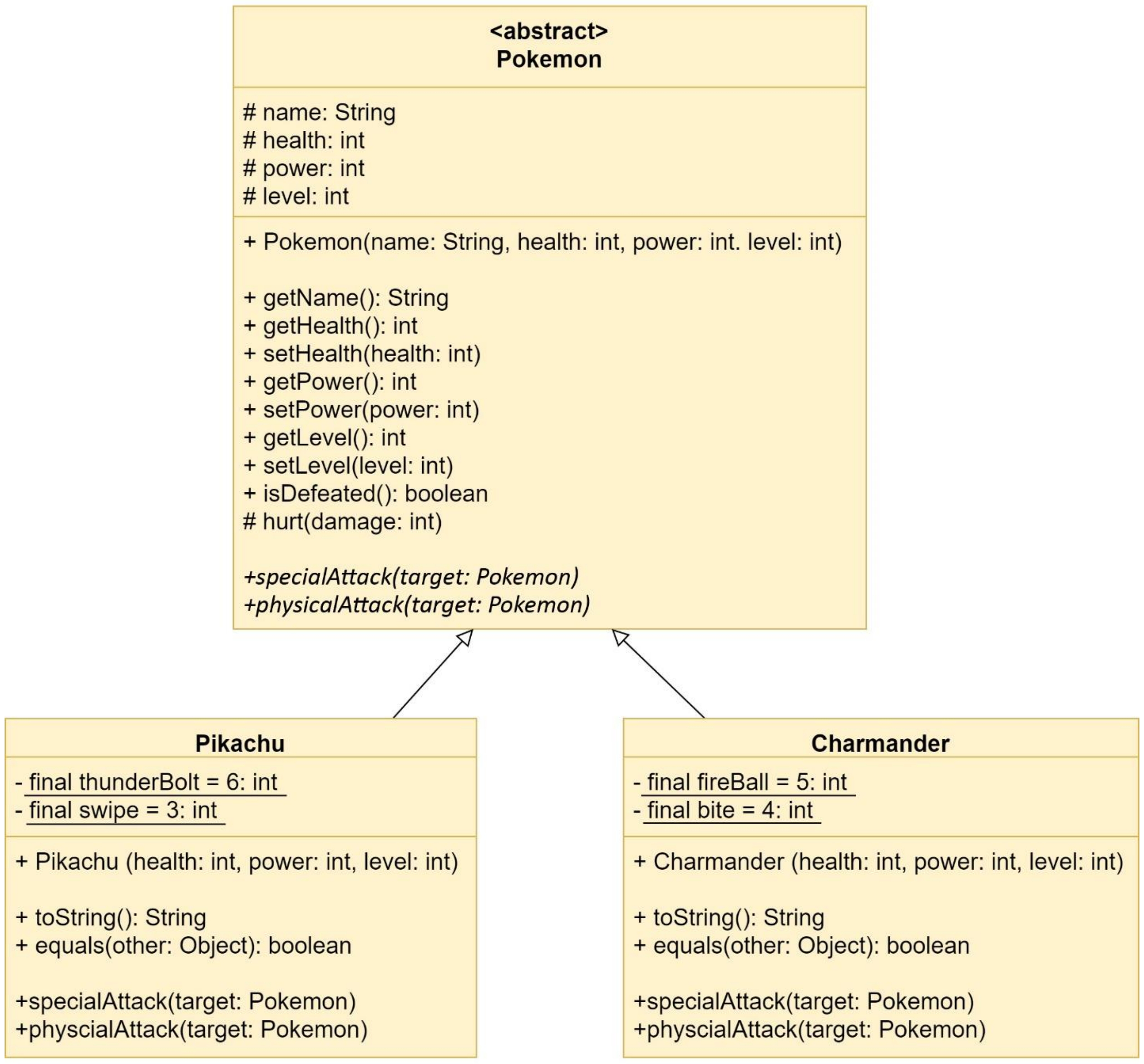
CompSci 251

Assignment 5

Due Oct. 15th, 2018

Program Specification

You will build a class inheritance structure for a small portion of the game Pokemon. You must implement the class hierarchy below. Your concrete classes must call the superclass constructor with the proper parameters. The # symbol in the UML diagram means it is protected, underlined means static, and italicized means abstract. The two italicized methods must be implemented in the subclasses.



<abstract>
Pokemon
# name: String
# health: int
# power: int
# level: int
+ Pokemon(name: String, health: int, power: int. level: int)
+ getName(): String
+ getHealth(): int
+ setHealth(health: int)
+ getPower(): int
+ setPower(power: int)
+ getLevel(): int
+ setLevel(level: int)
+ isDefeated(): boolean
# hurt(damage: int)
+specialAttack(target: Pokemon)
+physicalAttack(target: Pokemon)
Pikachu
- final thunderBolt = 6: int
- final swipe = 3: int
+ Pikachu (health: int, power: int, level: int)
+ toString(): String
+ equals(other: Object): boolean
+specialAttack(target: Pokemon)
+physcialAttack(target: Pokemon)
Charmander
- final fireBall = 5: int
- final bite = 4: int
+ Charmander (health: int, power: int, level: int)
+ toString(): String
+ equals(other: Object): boolean
+specialAttack(target: Pokemon)
+physcialAttack(target: Pokemon)

Requirements

- hurt(damage: int)
    - The hurt method is implemented in the superclass because it's implementation will not change from one type of Pokemon to the next. The hurt method takes an int argument (the damage to be dealt) that is to be subtracted from the health only if the argument passed in is positive and the pokemon isDefeated() returns false. If the value is negative or isDefeated returns true nothing happens, because if we subtracted this value from the health it would actually heal them or you are attacking something that is dead! If the attack is greater than the current health value, health is set to zero. Health should never be negative.

- specialAttack(target: Pokemon)
    - This attack method is abstract and must be overridden in each subclass as the attack is dependent on the type of Pokemon. The attack method should call the hurt method of the target Pokemon to decrease the health of the target Pokemon. The argument that is going to be passed will be either thunderBolt or fireBall. Both are of type int that represent the amount that will be taken from the target Pokemon's health for a single hurt. The amount of damage the attacking Pokemon's special attack does will be subtracted from their power after they attack. If the attacking Pokemon's power is not zero, but they do not have enough power for a normal special attack, their remaining power is used for the attack and their power is depleted. If their power is zero and specialAttack is called, you must call physicalAttack instead. This check can be done in specialAttack method. If either Pokemon isDefeated, meaning health is 0, nothing is to be done.

Example:

Charmander has power 7, special attack is 5.

Pikachu has health 15.

Charmander special attacks Pikachu, Pikachu health is 10, and Charmander power is 2.

Charmander special attacks Pikachu again, but its power is only 2 so it attacks for 2.

Pikachu health is 8, and Charmander power is 0.

Charmander special attacks Pikachu again, but its power is 0.

So Charmander does a physical attack instead, whose damage is 4.

Pikachu health is 4 now.

- physicalAttack(target: Pokemon)
    - This method is similar to specialAttack except that it does not need to check its power level for how much damage to deal, it always deals the same amount of damage to the target based on which Pokemon is attacking. If either Pokemon isDefeated, nothing happens.

- Setters
    - All setters/mutators for power, health, and level will set the instance variables if the argument passed in is greater than or equal to 0. Note that the name is read only, meaning it can not be changed so no setters for this.

- toString()
    - Prints out the contents of the Pokemon.

- equals(Object other)
    - The equals method takes an instance of Object which can be anything. We only have two types of Pokemon, but a Pokemon is only the same of another Pokemon if ALL of their instance variables are the same.

- Driver
    - Your main method should create a 4-element array of Pokemon. You should prompt the user four times for a name of the pokemon, health, and power. Each time, create a concrete instance of a Pokemon by calling the makePokemon method with the appropriate arguments. Add this Pokemon to the next index in the array only if the Pokemon was successfully created and if this Pokemon does not occur in the array already. There are two methods we require you to implement to help with creating/adding pokemon. The methods are

      static Pokemon makePokemon(name: String, health: int, power: int, level: int) { . . . }

      static boolean contains (Pokemon pokemon, Pokemon[ ] pokemons) { . . . }

      The makePokemon method should return the proper concrete subclass, determined by the name, or return null if the value of name is not Pikachu or Charmander (is case sensitive). This means every name of every Pokemon created will either be “Pikachu” or “Charmander”. Keep this in mind when calling the super constructors.

      The contains method should check if the passed Pokemon object exists in the array or not. Duplicates are not allowed. If a non-null, unique type of Pikachu or Charmander is created, it means it can be added to the array.

You will be provided with two helper methods, play and print. The play method will prompt the user three times to enter two integer numbers, entered on the same line and separated by a space. The first number is the Pokemon who will attack, the second number is the Pokemon to get attacked. The print method will print the content of the array.

Sample Output

I have added the comments to explain a few things, they will not show in your runs.
It is ok if your program crashes while taking invalid user input.

```
Enter name, health, power, and level for Pokemon#0: evee 200 10 5           //illegal
Enter name, health, power, and level for Pokemon#0: pikachu 100 10 10       //good
Enter name, health, power, and level for Pokemon#1: PIKAchu 300 20 20       //good
Enter name, health, power, and level for Pokemon#2: PiKaChu 300 20 20       //duplicate
Enter name, health, power, and level for Pokemon#2: charmander 150 12 12    //good
Enter name, health, power, and level for Pokemon#3: pikachu 100 10 10       //duplicate
Enter name, health, power, and level for Pokemon#3: p 3 3 3                 //illegal
Enter name, health, power, and level for Pokemon#3: charmander 200 20 15    //good

Pokemons before playing
0: Pikachu [name=Pikachu, health=100, power=10, level=10]
1: Pikachu [name=Pikachu, health=300, power=20, level=20]
2: Charmander [name=Charmander, health=150, power=12, level=12]
3: Charmander [name=Charmander, health=200, power=20, level=15]

# 0 Please enter the pokemons you want to play 0 to 3: -1 2                 //bad indices
# 0 Please enter the pokemons you want to play 0 to 3: 10 45                //bad indices
# 0 Please enter the pokemons you want to play 0 to 3: 1 2
# 1 Please enter the pokemons you want to play 0 to 3: 0 3
# 2 Please enter the pokemons you want to play 0 to 3: 0 1

Pokemons After playing
0: Pikachu [name=Pikachu, health=93, power=0, level=10]
1: Pikachu [name=Pikachu, health=292, power=14, level=20]
2: Charmander [name=Charmander, health=144, power=12, level=12]
3: Charmander [name=Charmander, health=194, power=20, level=15]
```