13.5 RMIT Practice together - Pokemons (inheritance)

Practice Exercises Objectives

This is a practice exercise designed to be done in class as a code along with a teacher. Consequently, some of the tests/feedbacks have not been fully tested and may be incorrect. This practice is not graded

You are tasked with writing three classes to represent objects that will be used as part of a Pokemon game.
The classes you will need to create are : Pokemon, BasicPokemon, and RarePokemon.
This assessment requires you to create these classes to represent the object types and enforce all the domain rules listed below:

Note:
You will need to read carefully the domain rules below and understand the (unit) tests to understand the use of the Pokemon, BasicPokemon, and RarePokemon classes.

You may add additional public methods.
You may (should?) to add additional private methods.
The unit tests rely on the documented specification of this class.
Therefore your classes will be assessed based on the documented constructors and methods.

Domain Rules (Pokemon)

You are required to code three types of pokemons: Pokemon, BasicPokemon & RarePokemon
It should not be possible to instantiate a Pokemon object.
Only BasicPokemon and RarePokemon objects can be instantiated.
Both Regular and RarePokemon must inherit from the Pokemon class.
You should use best practice when writing your code. For example class attributes should be

marked as private.

Pokemon Identifiers
Every pokemon has a pokemon identifier that is composed of two atomic identifiers: type and uniqueID.
These two identifiers are combined to create the pokemon identifier

Type
A type can be identified by a lowercase type identifier which can be only among: psychic, water, fire,electric
If a type has some uppercase letter then it should automatically be converted to an uppercase letter.
If an invalid type is provided then it should be changed to "n/a".

Pokemon ID
It can be between (1 - 1000).
ID number divisible by 42 or 73 are not allowed (they are dedicated to legendary pokemons that are not covered in this exercise).
If an invalid value is used for the uniqueID, then the uniqueID should be set to "0" to indicate an error has occurred.

Attack
All pokemons have an attack. Each pokemon type has a default attack.
Please see the rules for each type of pokemon to determine the attack used when creating a pokemon.

Trainer
A pokemon does not have a trainer at the time of its creation.
It is not possible to assign a trainer if a pokemon already has one.
A trainer can be cancelled only if the pokemon already has one.
Note that we do not record the name of the trainer, we only record have/not have trainers.

Domain Rules (Basic Pokemon)

Basic pokemons have a "kick" attack.
It should not be possible for the attack to be set to an invalid value.

Domain Rules (RarePokemon)

Attack
Rare pokemons have a "double kick" attack.

It should not be possible for the attack to be set to an invalid value.

Additional Features
Only rare pokemons of type electric have a super attack.
Only rare pokemons of type psychic have a ultimate attack

Summary of classes and required public methods

- Pokemon
  - public Pokemon() [usage optional]
  - public Pokemon(String, int, String)
  - public int getUniqueID() [*corrected return type*]
  - public String getType()
  - public String getIdentification()
  - public boolean hasTrainer()
  - public boolean setTrainer()
  - public boolean removeTrainer()
  - public boolean equals(Pokemon)
  - public String toString()
- BasicPokemon
  - public BasicPokemon(String, int)
- RarePokemon
  - public RarePokemon(String, int)
  - public String toString()

589850.4329690.qx3zqy7

LAB ACTIVITY | 13.5.1: RMIT Practice together - Pokemons (inheritance) | 0 / 34

Current file: LabProgram.java | Load default template...

```
1  public class LabProgram {
2      public static void main(String[] args) {
3          /**
4           * Sample code has been provided as starting point to test your
5           * implementation.
6           *
7           * IMPORTANT: This code will not compile until you write the classe
8           * instantiated here.
9           * You may modify this code for faster testing purposes and explora
10          * Adding undocumented public methods to your Pokemon, BasicPokemon
11          * may mean that the unit tests fail as our testing code will be un
12          * methods and therefore cannot call those methods.
13          */
14
15         // UNCOMMENT THE CODE BELOW ONCE YOU HAVE WRITTEN YOUR CODE TO TEST
16         // IMPLEMENTATION
17
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **LabProgram.java** (Your program)

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```

Trouble with lab?