Worksheet 3

Object design

In this section we will focus on class design to model objects for a game.

Pokemons

Suppose we want to implement a pokemon game in Java. You probably already know the game but just in case, know that Pokemons are creatures which develop and grow: they gain experience, or "XP", which accumulate until they have enough to gain a new "level" (a baby pokemon starts at level 1 and can climb all the way to level 100, where they remain forever — but in earlier versions of the game, some glitches allowed to encounter pokemons of a level higher than 100, but always strictly inferior to 255... if you managed to catch one and won a fight with them they would immediately revert to level 100 or 1, different accounts by different people gave different, hard to know which one was true if any, given how easily a level 248 pokemon would effortlessly distroy all your team of pokemons but I digress).

How would you describe one particular species, say Lapras?

- create a new class in an empty file named `Lapras.java`, to represent the species itself, from which particular individual pokemons, the *instances* of the class will be made
- think about the fields you need a pokemon to have: how would you name them, what types should they be?
- implement a first constructor for the class

Each individual has a name which is by default the name of its species, but which can be altered later in the game

- modify the fields in your `Lapras` class to support the name and the constructor accordingly

New individuals of a given species are created in the game + as babies when a pokemon cracks its egg + as a wild pokemon of a given level, usually randomized in a range depending on the geographical zone within the game

- modify the existing constructor accordingly and add a new one to support this new possibility

To gain experience pokemons fight, by throwing attacks at each others ("moves" in Pokemon parlance) which can directly hurt the opponent or have various effects on the game such as modifying their own status or the opponent's in good or in bad (there are several different ways to boost a pokemon or to reduce its abilities, to make it sleep, or burn, or be poisoned...). We won't go into to much detail about implementing attacks at first.

- create a class `Move` in a new `Move.java` file. Let's say for now that an attack is defined by its name, the damages it does and a possible effect, which will only be printed to the player for now. Add the corresponding fields and constructors

It should be noted that as they grow pokemon acquire new attacks based on their level (each species spontaneously learns each move it can learn when it reaches a specific level: for instance, all pokemons of the same species might learn say `growl` at level 7, `byte` at level 15 and `heal` at level 27). But at any time, a pokemon cannot know more than 4 moves (it can know less and often starts with only 1 or 2 moves).

- how could we represent such a piece of data (the levels at which a given species will try and learn new moves)?
- is that a trait tied to each individual or to the species itself? As a consequence, would you make this property in the class `static` or not?
- implement a list of attacks for `Lapras` to learn based on the game data

Oh, and there are elemental types too: water - fire - plant — ground - rock - eletrik (yeah they spelt it that way, go figure...), psy... and a bunch of others which you can probably look online. There is a intricate algebra of efficiency/resistence a bit like rock/paper/scissors but on steroids. You can probably look it up too, but it changed slightly across the versions so don't worry too much about it, just pick the types you want to support and try to get some more or less plausible efficiencies (if water doesn't extinguish fire which burns plant which absorb water, I don't wanna play your game...). We will see in next class a good feature of Java to represent (pokemon) types, so you can just use `String` for now.

- make a new class in a separate file to handle types. Again, don't worry too much about their representation, but just focus on exposing the required logic: how efficient an attack of a given type is on a pokemon based on its type

Each move has an associated type, and each pokemon species also has at least one type, and can have up to two. They needn't always align — most pokemons learn "normal" type moves, and can sometimes learn a couple moves out of their type.

- `Lapras` is type "water", so modify the class accordingly — again, ask

yourself whether this should be a property of each individual or of the species as a whole and pick the field modifiers accordingly
- modify the `Move` class too to add a type

Ok, so far, we have done only composition by adding field of a given class to another class, but it's high time we enriched our game with other species (there are hundreds of them after all). Let's say we want to have `Pikachu` and `Raichu`.

- we are certainly not going to reimplement everything we did for `Lapras` into each of the new classes: let's take all the properties they should share into a mother `Pokemon` class which all pokemon species should *extend*. Create a new file for the class accordingly
- would it actually make sense to instanciate a `Pokemon` out of any particular species? So what keyword should you use on that class?
- move all the code you want to share between your species into the new class
- which common property or behaviour custom to each species should you leave unimplemented? How would you do so?

Finally pokemons "evolve": a given species will spontaneously turn into another one when certain conditions are met. Most of the time, reaching a given level is enough (and then, if evolution is blocked it will retry at each new level), but sometimes a particular object (an evolution stone) is required as is the case for a `Pikachu` to become a `Raichu`.

- update the `Pokemon` class to take this new ability into account
- update the implementations of `Lapras`, `Pikachu` and `Raichu` to reflect the fact that both `Lapras` and `Raichu` can't evolve, but that `Pikachu` can turn into a `Raichu` when given a thunder stone.
- so in a sense, an individual pokemon can switch species (when evolving), even though it keeps all the moves it already knows. What would be the best way to represent this? There's not necessarily a best way to do so, maybe we could add a new constructor to the evolved species taking an instance of the base species as argument, or maybe we should entirely decorelate individuals from their species, and use composition: maybe the species of a pokemon should have been only one field in a pokemon among others?

Packaging

Now, the model described above will be used to implement a version of the game itself, but it could also be extremely useful to create tools to help players learn about the game, optimize their strategies, so that it could make sense to expose the data encoded within the library as a web API.

For that reason, we are going to create a separate package for the pokemon library encoding the game data, and other packages for the game and the API.

- create the corresponding structure in the directory where you work for this practice
- move the files you've already created to the right place.

Let's start implementing tools

- make a new executable program (`Main` class, `main` entry point etc.) leveraging the library to create a couple pokemons and make them fight, ideally with some interaction from the user if you want your program to look like a game (choose a move, see the result, handle damage...)
- make another executable to query information about the various species, taking simple queries as input and displaying the relevant information extracted from the library
- in the process, you should notice everything that you need to expose and modify the visibility of the various fields you've introduced in your code accordingly