Hoja Trabajo 6
Algoritmos y Estructuras de Datos
Adrian Arimany - 211063

The three main structures that were implemented in the program are:

1. HashMap
2. LinkedHashMap
3. TreeMap

Although our default choice was HashMap, our program can initialize with any of these three options.

Explanation:

HashMap:
It offers fast (average constant-time) performance for basic operations like get and put. However, it doesn't preserve any order of keys, which is acceptable if ordering isn't needed.

LinkedHashMap:
This implementation maintains insertion order. It's useful when you want the Pokémon to appear in the order they were added, which can improve the predictability of display and iteration.

TreeMap:
TreeMap stores keys in a sorted order (either natural ordering or via a provided comparator). We use it when we need our data (such as Pokémon sorted by their "Type1" attribute) to be automatically ordered.

The data structure:

First we defined our map as Map<String, PokemonRecord> (please check the things to consider below). Where take the row from the file as a String, and we label each column as a PokemonRecord. This map from the CSV file to the pokemon's attribute. Here we have the following methods for each pokemon:

addPokemon(String name, String Ability): Boolean
Adds a new pokemon by its name and its type.

If the name exists in the csv, then returns false, otherwise returns true and adds it to the csv file.

searchByName(string nameQuery): List<PokemonRecord>
Look for the Pokemon that has the attribute with the same name. Though the data structure is a List, even so there aren't supposed to be any duplicate pokemons with the same name, we kept it as a list in the odd case that someone decides to manually insert a pokemon directly in the csv file. This way our program can catch any of such oddities.

searchByAbility(String abilityQuery): List<PokemonRecord>
Look for the Pokemons that have the attribute with the same ability. This makes obvious sense to be a List, because there will be more than one pokemon that has the same ability. What it will return is the Name, Type1, and the Ability.

getAllPokemon(): String
Returns all the pokemons by Name and Type1. (again check things to consider)

loadFrom(String filePath): void
The current program only works with a CSV file. What this method does is it checks the CSV file, it looks for the headers, and only stores the headers Name, Type1, Attributes. It then stores these columns in a new Map, which will be using one of the three JCF mentioned above.

saveTo(String filePath): void
This method only exists so that when the user adds a new pokemon, that pokemon is saved in the csv file. So that the user can store these newly added pokemons. (Although this wasn't part of the assignment).

Overall, I believe that the best data structure to implement is a List<PokemonRecord>, which takes the key of the Map as its indicator. The main reason for this is because the Data File Type is a CSV, which is not flexible with the use of commas (","), unless we use a legacy package to make CSV files workable with commas. If for example, we used a JSON file, then we could have used maps in these methods, but that is because a JSON file is better suited to use MAPS.

We used CSV file over JSON file, merely because the file that we extracted the Pokemon Data from has the pokemon data as a csv file as its only downloadable option. We didn't implement the legacy CSV, because we haven't learned how to use that in class.

Things to consider,

In the instructions, it was asked to use Name as our key in the Map. But this is counterproductive, because in our data set, there is a column called Name, which would cause confusion when trying to reference the pokemon. Especially when you had to map the Pokemon’s ability. So for that reason our Key is called PokemonRecord, this way we can better generalize how we map the csv.

Also I found a rather contradictory statement in the instructions. Because in Operation (2) it says that we need to show the attributes for that pokemon, although it doesn't quite specify which attributes. Then in Operation (3) and Operation (4) now the program is meant only to show the Name, and its Type1. So I assumed that Operations (3) and Operation (4) supersedes Operation (2).