CSCI3100: Software Engineering

Assignment 3

March 27, 2025

`Due date: 11 Apr 2025 Total marks: 110`

Revision	Date	Description
1.0.0	27 Mar	Initial release
1.0.1	28 Mar	Package: Fixed implementation Doc: Fixed typo: PokemanTrainingData -> PokemanTrainingStats
1.0.2	7 Apr	Added explanations and constraints for the questions

0.1 Questions

1. 1. (*)Redesign the `update_stats()` function so that it accepts, instead of 5 integers, an instance of a data class called `PokemanTrainingStats` containing the 5 attributes. The class `PokemanTrainingStats` should be placed in `pokeman.py`. All the attributes of `PokemanTrainingStats` can be public. Modify the existing code. [20 pt]
   2. What are the two benefits of passing a data class object instead of a tuple of integers to `update_stats()`? Explain in at most two sentences. [10 pt]
   3. After adding `PokemanTrainingStats` in `pokeman.py`, `training.py` will need to refer to `pokeman.py`. How did you resolve the circular reference:

   - `pokeman.py` imports `training.py` and
   - `training.py` imports `pokeman.py`

   Explain in one sentence. [5 pt]

4. How can the circular reference be avoided in the first place? Explain in one sentence. [5 pt]

Remarks: A circular reference between `pokeman.py` and `training.py` will exist if type hint is given in `training.py` for `PokemanTrainingStats` because `pokeman.py` refers to the data structures in `training.py`, and vice versa. There are some ways to avoid the circular reference. Some of the solutions are better in view of software engineering, whereas some other ways are super developer-friendly that make use of the dynamic nature of Python. All solutions are accepted.

Part 4 expects a short description of the best practice for avoiding circular references. No matter how you have avoided circular reference in part 1, please write the best solution under this circumstance.

`SoftEngPokeman` is not related to `Pokeman`. It does not need to be modified.

2. Regarding the observer pattern employed in the design:

    1. (*)The current design assumes the `Trainable` has some attributes such as `hp`, `level`, and `name`. These assumptions make `Trainable` not quite generic. How can it be more generic, and thus allow classes implementing it to have their own attributes and operations? That is,
        1. How can the function signature of `update_stats()` be modified to accept the attributes to be updated without specifying them explicitly as parameters?
        2. How can `get_level()` and `get_name()` be unified as a single function, letting the user get whatever attributes they want to see?
        3. What is the disadvantage of this generalisation approach?

        Modify the existing code. [20 pt]

    Remarks: This question asks how to make `Trainable` as generic as possible. The code using `Trainable` needs not be generic.

    `SoftEngPokeman` is not related to `Pokeman`. It does not need to be modified.

    In the previous Blackboard announcement, we mentioned `TrainingData` for Q2. Please ignore it. Our apologies.

3. (*)Without modifying any existing source code, how can `SoftEngPokeman` be trained with `PokemanGym` at runtime? Implement all the necessary code in a new file named `softeng_pokeman_trainer.py`. Demonstrate training an object of SoftEngPokeman to the max level in `main.py`. [30 pt]

    Rules:

    1. `SoftEngPokeman` has to be processed by `train_pokeman()`, or `finished_training()` in `PokemanGym` at runtime.

2. No compile-time direct call to `finished_training()` on the `SoftEngPokeman` object.
3. May need to use duck-typing.
4. `SoftEngPokeman` cannot be inherited from `Pokeman`.
5. Cannot copy methods from `Pokeman` to `SoftEngPokeman` (no modification to existing source code after all)

Remarks: Rule 1 means the statistics of the instance of `SoftEngPokeman` should be directly updated via either `train_pokeman()`, or `finished_training()` defined in `PokemanGym`, or both of these two functions.

Training an instance of a subclass of `SoftEngPokeman` is not accepted.

It is not acceptable to create a new class similar to PokemanGym that have methods `train_pokeman()` and `finished_training()`.

It is not acceptable to develop some types of "adapter" that includes a `SoftEngPokeman` and then updates the statistics of the `SoftEngPokeman` through the adapter. (However, your efforts to apply what you have learned are greatly appreciated .)

How to solve this question? We do not need to use any advanced features of Python. It is accepted if you know how to use them, though. The idea is in fact very simple:

1. Since `train_pokeman()` accepts `Pokeman` only, we cannot use it to train `SoftEngPokeman` instances directly. Then, is it possible to train an `Pokeman`, but update the statistics of an `SoftEngPokeman` instead?
2. It seems it is possible, because `finished_training()` does not check the type of the argument. As long as the argument is a concrete `Trainable`, or have the function `update_stats()` (duck-typing), its statistics can be updated.
3. Note that the observer pattern in the code can be divided into two steps:
   1. The "observed" notifies the observer after it has been trained
   2. The observer updates the statistics of the "observed" after getting the notification

   Step 1 cannot be "hacked". It must be a `Pokeman` that notifies the observer. What if we could, somehow, set the identity of the "observed" from a `Pokeman` to a `SoftEngPokeman` before step 2?

4. (*)Write the code that uses and extends the existing code, and can set up the following scenario: [20 pt]

   - Create a `Pokeman` object with the name "ChuKaPi".
   - Train `ChuKaPi` to the max level.
   - Create a `SoftEngPokeman` object with the name "Kei".
   - Train `Kei` to the max level.

- Implement code that will allow `Kei` to battle `ChuKaPi` (and vice versa) in `BattleSystem`.
- Create a `BattleSystem` object.
- Run the `battle()` function.
- Print the result of the battle.

Remarks: Easy question. We expect making `Pokeman` and `SoftEngPokeman` compatible with the `BattleSystem` only. You can define any battle rules to make it more interesting.

Remarks: Don't worry about the expected output formats. We are checking the code. You can provide explanations for your answers in README.md to help with our understanding.

Questions marked with (*) requires coding. Please follow this procedure:

```
# 1. Prepare the directory for each of the question
# solution_<n>_src is for Question <n>, e.g. solution_1_src is for Question 1
cd asgn_3_package
cp -r question_src solution_1_src
cp -r question_src solution_2_src
cp -r question_src solution_3_src
cp -r question_src solution_4_src

# 2. Work inside the corresponding directory for each question

# 3. Write all other non-programming answers in asgn_3_package/README.md
```

0.2 Submission

1. What to submit:
    1. Your answers written in a Word or PDF document, if any
    2. The associated source code files, if any
2. Pack everything in a zip file named ".zip". E.g. if your student ID is `1234567890`, name the zip file as `1234567890.zip`
3. Submit the zip file to Blackboard