CSCI3100: Software Engineering

Assignment 3

March 27, 2025

```
Due date: 7 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

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]

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]

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`.

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.

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