Abgabe: 15.12.2023

Aufgabe 1

In dieser Aufgabe beschäftigen wir uns mit Objektorientierung in Python. Der Fokus liegt auf der Implementierung einer Klasse, dabei nutzen wir insbesondere auch Magic Methods.

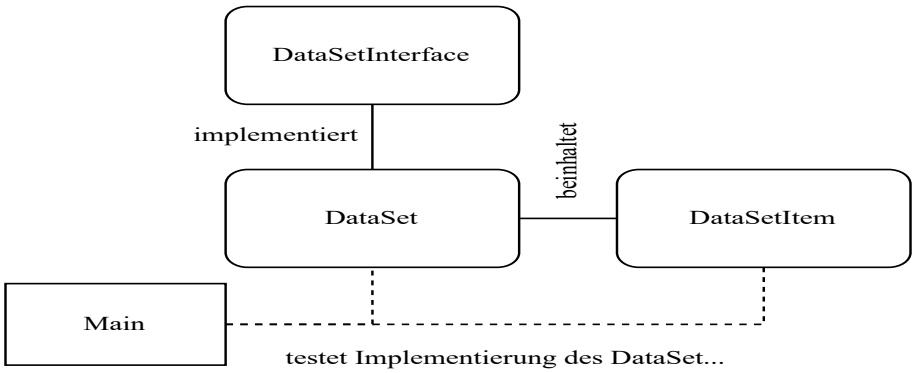

Figure 1: *
Abbildung 1: Darstellung der Klassenbeziehungen.

Ein Datensatz besteht aus mehreren Daten. Ein einzelnes Datum wird durch ein Objekt der Klasse `DataSetItem` repräsentiert. Jedes Datum hat einen Namen (Zeichenkette), eine ID (Zahl) und beliebigen Inhalt.

Nun sollen mehrere Daten, also Objekte vom Typ `DataSetItem`, in einem Datensatz zusammengefasst werden. Sie haben sich schon auf eine Schnittstelle und die benötigten Operationen, die ein Datensatz unterstützen muss, geeinigt. Es gibt eine Klasse `DataSetInterface`, die die Schnittstelle definiert und die Operationen jedes Datensatzes angibt. Bisher fehlt jedoch noch die Implementierung eines Datensatzes mit allen Operationen.

Implementieren Sie eine Klasse `DataSet` als eine Unterklasse von `DataSetInterface`.

Es gibt drei Dateien: `dataset.py`, `main.py` und `implementation.py`[1] In der `dataset.py` befinden sich die Klassen `DataSetInterface` und `DataSetItem`, in der Datei `implementation.py` muss die Klasse `DataSet` implementiert werden. Die Datei `main.py` nutzt die Klassen `DataSet` und `DataSetItem` aus den jeweiligen Dateien und testet die Schnittstelle und Operationen von `DataSetInterface`.

Bei der Klasse `DataSet` sind insbesondere folgende Methoden zu implementieren. Die genaue Spezifikation finden Sie in der Datei `dataset.py`:

1. `__setitem__(self, name, id_content)`: Hinzufügen eines Datums mit Name, ID und Inhalt.
2. `__iadd__(self, item)`: Hinzufügen eines `DataSetItem`.
3. `__delitem__(self, name)`: Löschen eines Datums basierend auf dem Namen. Der Name eines Datums ist ein eindeutiger Schlüssel und darf nur einmal pro Datensatz vorkommen.
4. `__contains__(self, name)`: Prüfen, ob ein Datum mit diesem Namen im Datensatz vorhanden ist.
5. `__getitem__(self, name)`: Abrufen des Datums über seinen Namen.
6. `__and__(self, dataset)`: Schnittmenge zweier Datensätze bestimmen und als neuen Datensatz zurückgeben.
7. `__or__(self, dataset)`: Vereinigung zweier Datensätze bestimmen und als neuen Datensatz zurückgeben.

[1] https://github.com/ITI-WerkzeugeWissArbeiten/Git-2024-25/tree/main/code

8. `__iter__(self)`: Iteration über alle Daten des Datensatzes (es ist möglich, eine Sortierung für die Reihenfolge der Iteration anzugeben).
9. `filtered_iterate(self, filter)`: Gefilterte Iteration über einen Datensatz. Eine Lambda-Funktion mit den Parametern Name und ID dient als Filter.
10. `__len__(self)`: Abrufen der Anzahl der Daten in einem Datensatz.

Abgabe

Programmieren Sie die Klasse `DataSet` in der Datei `implementation.py` zur Lösung der oben beschriebenen Aufgabe im VPL. Sie können auch direkt auf Ihrem Computer programmieren. Dazu finden Sie alle drei benötigten Dateien zum Download im Moodle.

Das VPL nutzt denselben Code. Die Datei `main.py` wurde jedoch um weitere Testfälle und Überprüfungen erweitert. Diese Überprüfungen dienen dazu, sicherzustellen, dass Sie die richtigen Klassen nutzen.