Final Project

College of DuPage

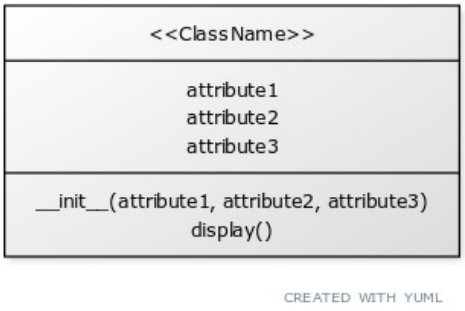
Course Title:	Academic Semester:	Last Update:
Intro to Python	Spring 2022	April 17, 2022

Part 5: Creating a Class for your Data

In Part 4, you wrote a bare-bones interface for entering and displaying your data. However, it was awkward to store your data in a multidimensional list, and will make it difficult to add more features later.

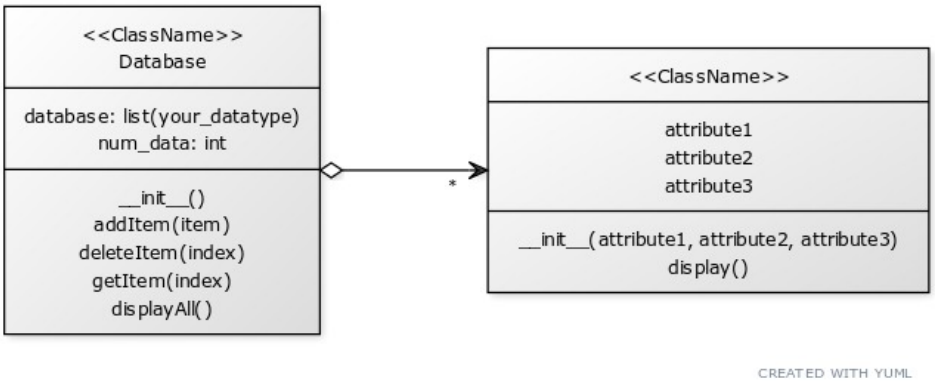
Create two Python files in the same directory as your `databaseio.py`.

The first file should be named `[your_datatype].py`, where `[your_datatype]` is meant to be replaced by a discriptive name for your datatype. For example, in my NFL Players example, I will name this file `nfl_player.py`. This file should contain a class (named appropriately), which matches the following UML diagram.



- `attribute1`, `attribute2`, and `attribute3` are your 3 attributes, chosen back in Part 1;
- `__init__(self, attribute1, attribute2, attribute3)` is the constructor, which sets `self.attribute1 = attribute1`, `self.attribute2 = attribute2`, and `self.attribute3 = attribute3`
- `display(self)` prints `self.attribute1`, `self.attribute2`, and `self.attribute3` by calling `databaseio.print_data()`

The second file should be named `[your_datatype]_database.py`, where `[your_datatype]` is meant to be replaced by a discriptive name for your datatype. For example, in my NFL Players example, I will name this file `nfl_player_database.py`. This file should contain another class (named `[ClassName]Database`, where `[ClassName]` is the name you chose for the previous class). This new class should match the following UML diagram.



- `database` is a Python list of `[your_datatype].[ClassName]` objects and `num_data` is an integer tracking the number of entries in `database`;
- `__init__(self)` is a constructor, which sets `database = []` and `num_data = 0`.
- `addItem(self, item)` appends an item (of type [your_datatype]) to the `self.database` list and increments the counter `self.num_data` by +1;
- `deleteItem(self, index)` removes the entry at index `index` from `self.database` (hint: use Python's `[].pop(i)` method);
- `getItem(self, index)` returns the `[your_datatype].[ClassName]` object at `self.database[index]`;
- `displayAll(self)` prints every item in `self.database` using their `display()` method (hint: one way is to loop over index i and use `self.getItem(i).display()`).

Finally, create a new file `main_pt5.py` which modifies the contents of the `main_pt4.py` file from the last assignment to use these new classes and methods. You should be able to reuse a lot of code from your main execution loop, and should continue to use the `databasio` module, wherever appropriate.

- Instead of starting with an empty list for your database, start by creating a new `[your_datatype]_database.[ClassName]Database` object;
- Whenver the user adds an item to the database, first call `databasio.read_data()` to get input from the user. Then create a new `[your_datatype].[ClassName]` object and add it to your database by using `your_database.[ClassName]Database.addItem(item)`;
- Display the contents of your database by calling `your_database.[ClassName]Database.displayAll()`;
- Delete an item from your database by calling `your_database.[ClassName]Database.deleteItem(index)`.

Deliverable:
For this part, send me the following deliverables.

- Resend me your `databaseio.py` file, which should still be unmodified from Part 3.
- Send me your 2 class files `[your_datatype].py` and `[your_datatype]_database.py`.
- Send me your updated `main_pt5.py` file.
- Send me the output from running `main_pt5.py` in `main_pt5_test.txt`, which should not have changed much (if at all) from Part 4.

Example:
For my NFL players example, I would turn in

- `databaseio.py`,
- `nfl_player.py` (containing the class `NFLPlayer`),
- `nfl_player_database.py` (containing the class `NFLPlayerDatabase`),
- `main_pt5.py`, and
- `main_pt5_test.txt`, whose output should look identical as in Part 4, but you should reproduce this file again using main_pt5.py, to make sure you can still perform all the same operations.