ACIT 2515 – Object Oriented Programming – Assignment 3

Instructor	Mike Mulder (mmulder10@bcit.ca)
Total Marks	30
Due Dates	Wednesday, April 10th, 2019 by midnight

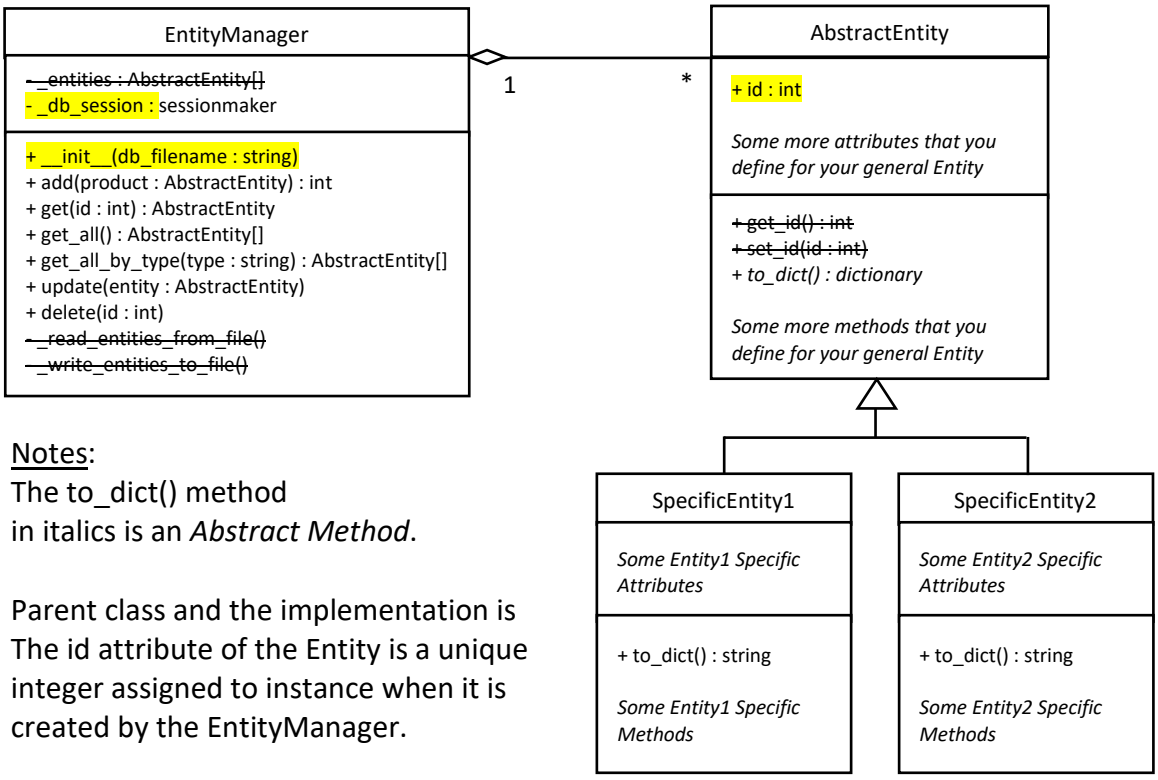
You will be continuing this with the same code and partner as Assignments 1 and 2.

Goals

- To persist (i.e., store) your entities to a Sqlite database.
- To map your entities to database tables using the SQLAlchemy ORM.
- To create a Graphical User Interface (GUI) in Tkinter that integrates with your RESTful API using the Python requests package. The GUI will allow users to create, update, delete and query the entities stored in the database.

Overview

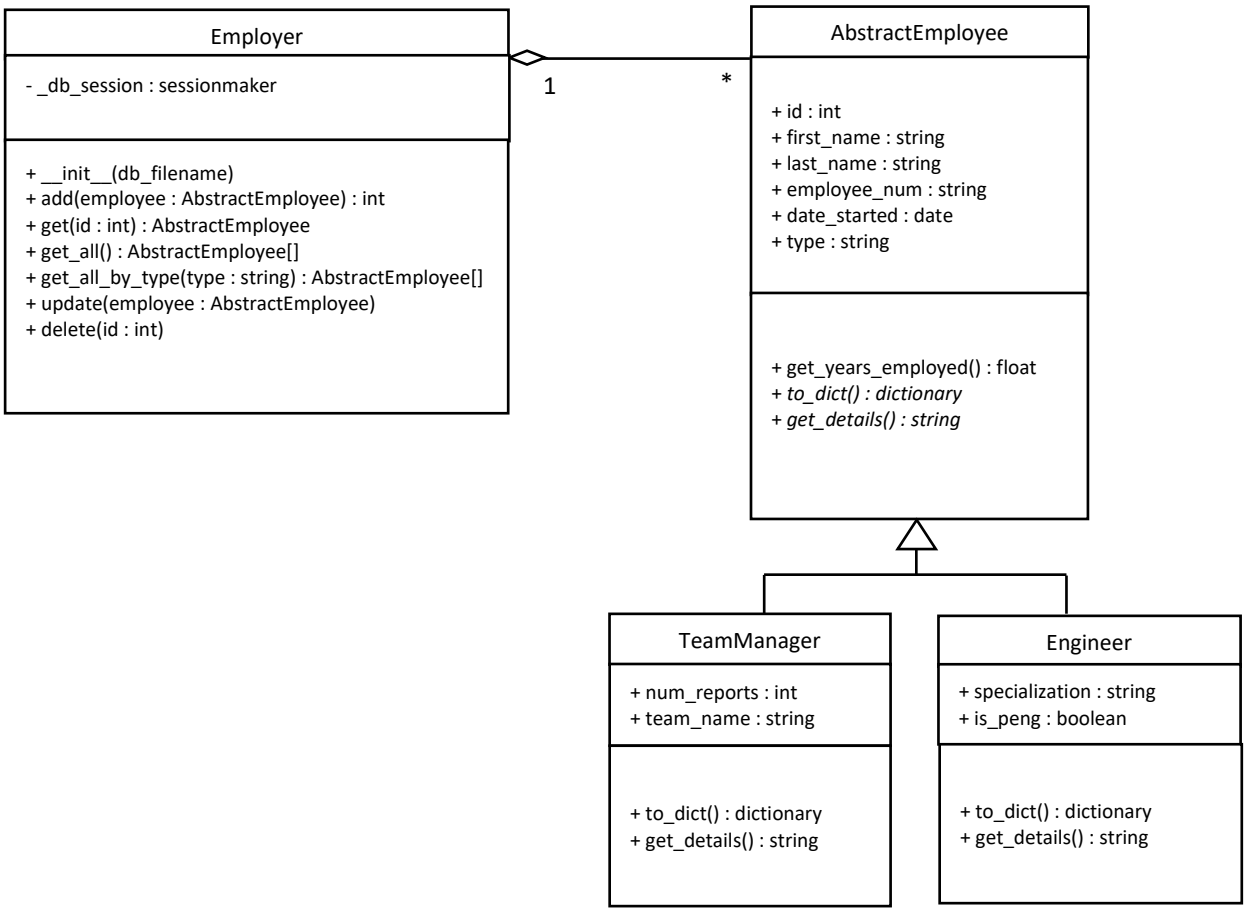
For Assignment 2 you implemented the following UML design. For the first part of Assignment 3, you will need to update your UML (as indicated below) to reflect the conversion of your entities to SQLAlchemy declaratives and your datastore to a Sqlite database. See the example on the next page for how that might look. Remember that attributes on SQLAlchemy declaratives are typically public.



Notes:
The to_dict() method
in italics is an *Abstract Method*.

Parent class and the implementation is
The id attribute of the Entity is a unique integer assigned to instance when it is created by the EntityManager.

Here is a representative example of what your design might look like.



Note the following:

- The list of entities is no longer needed as they are stored in the database.
- The name of the Sqlite database is now passed into the constructor of the Employer class.
- The filepath of the json file from Assignment 2 is now the _db_session of the Sqlite database.
- The _read_entities_from_file and _write_entities_to_file methods are no longer needed.
- All entity attributes are public (as per SQLAlchemy) so getter and setter methods are no longer required.
- Note that constructor methods are required for the entity classes (i.e., Employee, TeamManager and Engineer) but not reflected in the UML above.

Design (4 marks)

Update your UML in accordance with the changes described above.

Database Persistence (12 marks)

For this assignment you will be storing your entity records in a single table in a Sqlite database. Design your database table such that it can store the attributes for both entities. Some design decisions:

- Sizes for your string (varchar) columns. Make sure they are big enough to hold your data. And make sure the parameter validation in your entity class constructors reject values that are too big.
- Mandatory columns (i.e., not nullable). Note that the attributes specific to your child entity classes must be nullable because they will not be populated for the other entity type. If they mandatory, that must be enforced through parameter validation – both in the constructor in the SQL Alchemy declarative and the GUI.

You need to create the following files:

- create_tables.py – Create your database table will all the required columns to support both your specific entity types, including a column to hold the type.
- drop_tables.py – Drops your table.

You will be using SQLAlchemy as the ORM to interact with your database table.

You need to create the following file:

- base.py – You may copy this from Tutorial 2 or Labs 8/9.

You need to update the following files:

- Your specific entity classes. These must be re-written as SQLAlchemy declaratives. They should have all the same helper methods as before <u>including</u> a constructor with parameter validation.
- Your entity manager class. This must be re-written to use SQLAlchemy to create the connection to your Sqlite database file and all methods refactored to create, update, delete and query the database using SQLAlchemy.
    - All previous public methods remain, just the implementation changes
    - All previous parameter validation and exceptions must remain

Use the example in Tutorial 2 and Lab 9 as guidance for this part of the assignment.

Unit Testing (4 marks)

Update the unit test for your entity manager to work with your refactored entity manager class.

There should NOT be significant changes to the unit tests, since your public interface is largely unchanged, except:

- Your setUp method should create a test sqlite database (it should have a different name than the one used in your application).
- The tearDown method should delete the test sqlite database.

Use the unit test from Tutorial 2 as an example of the setUp and tearDown.

You do NOT need unit tests for the specific entity classes for this assignment.

Graphical User Interface (10 marks)

Create a GUI for your entities that supports the following:

- Displays a summary list of the specific entities (stored in your database)
    - Toggle the list of specific entities between each of the two types
- Displays the full details of a single entity upon selection from the list above (both types supported)
- Add a new entity of each type
- Update an existing entity (both types supported)
- Delete an existing entity (both types supported)

Make sure the GUI displays errors returned from the backend in a user friendly manner. It should not just display the error code (i.e., 200, 400, 404) to the user. It should display messages like:

- Employee successfully added
- Employee could not be added because data is missing or invalid
- Etc.

The GUI must call the API endpoints of your RESTful API for each of the above actions using the *requests* Python package. It must NOT integrate directly with your entity manager – all interaction between the GUI and the backend should be through the RESTful API.

Grading on the GUI will be on both the functionality AND the usability. If GUI is unorganized or difficult to use, you will lose marks.

Note that to run and test your application, both the RESTful API main application (i.e., employer_api.py) and the GUI main application (i.e., employer_gui.py) must be running at the same time.

Grading Summary

Design Updates • UML Updates (4marks)	4 marks
Database Persistence Updates	12 marks

• Database create/drop scripts (2 marks) • SQLAlchemy Declaratives for the Specific Entity classes (4 marks) • Refactored Entity Manager class (6 marks) No marks for this section if the RESTful API application no longer runs.	
Unit Test Updates • EntityManager test updates for SQLAlchemy declaratives (2 marks) • EntityManager test updates for test Sqlite DB creation and deletion (2 marks) No marks for this section if the unit test does not run without errors or unit test failures.	4 marks
GUI • 5 GUI views (10 marks) No marks for this section if the GUI application does not run.	10 marks
Total	**30 marks**

Marks will be subtracted for violations of best practices covered so far in this course (i.e., naming, DocString, constants for magic numbers, etc).

Submission

Submit the following in a zipfile called **Assignment3.zip** and upload to the dropbox in D2L (Activities -> Assignments -> Assignment 3):

- UML Design in PDF format (**uml_design.pdf**)
- All the code necessary to run your two applications, including the two main applications:
  - RESTful API module (i.e., employer_api.py)
  - GUI module (i.e., employer_gui.py)
- Unit test class (i.e., test_employer.py)

The submission is due Wednesday, April 10th at midnight for all sets. No late submissions will be accepted.