ACIT 2515 – Object Oriented Programming – Assignment 2

Instructor	Mike Mulder (mmulder10@bcit.ca)
Total Marks	30
Due Dates	Friday, March 22, 2019 by midnight

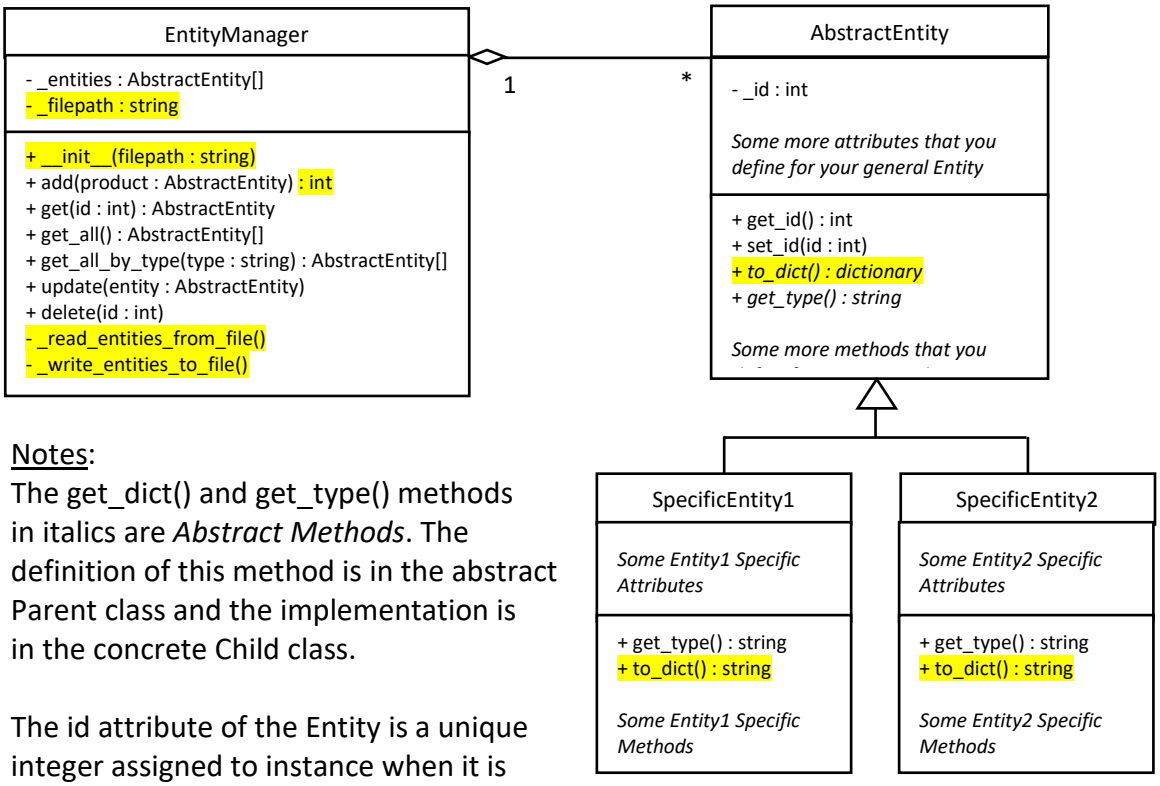
You will be continuing this with the same code and partner as Assignment 1.

Goals

- To build a RESTful API using JSON and Python Flask
- To persist your entity data using JSON

Overview

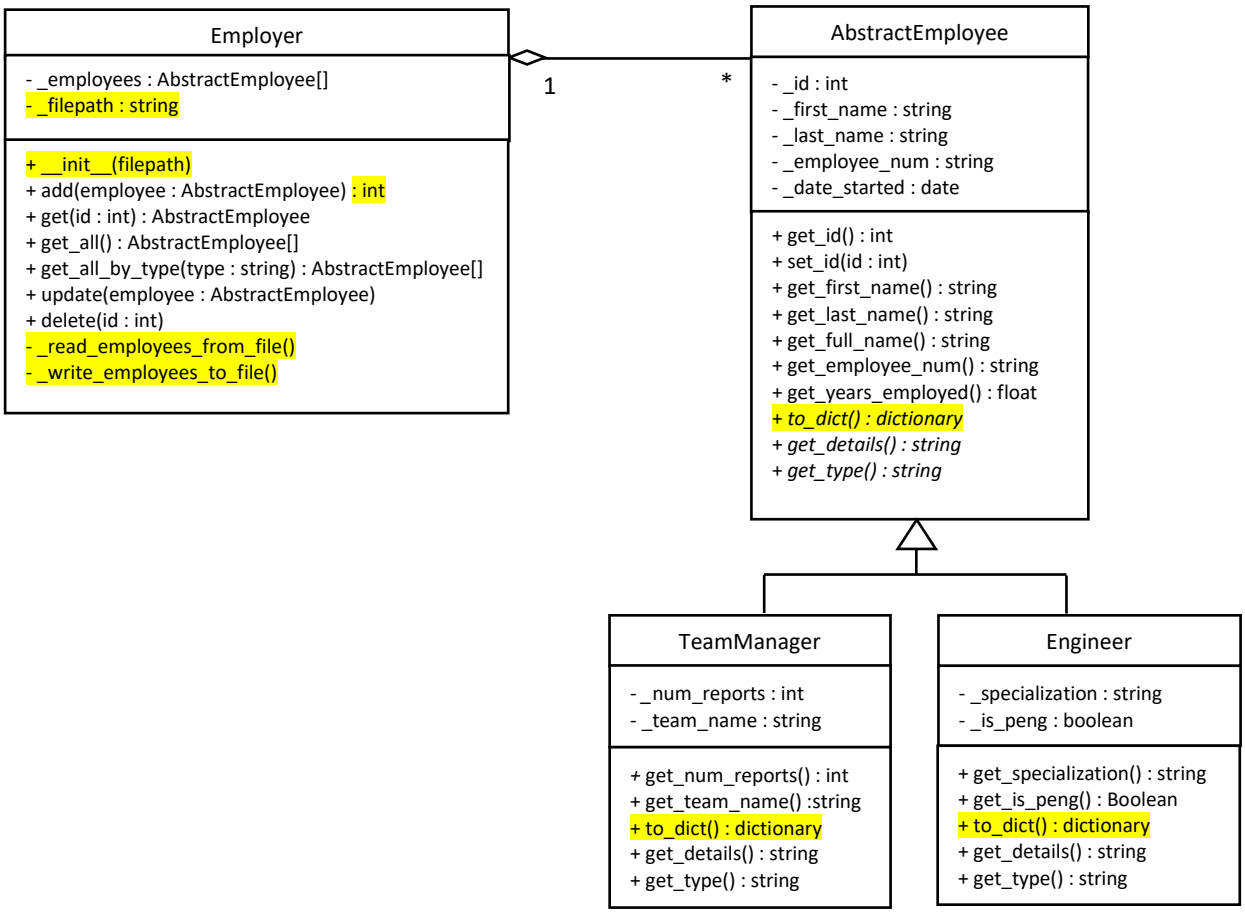
For Assignment 1 you implemented the following UML design. For the first part of Assignment 2, you will be modifying your code slightly with the changes highlighted in yellow.



Notes:
The get_dict() and get_type() methods in italics are *Abstract Methods*. The definition of this method is in the abstract Parent class and the implementation is in the concrete Child class.

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.


Employer
- _employees : AbstractEmployee[]
- _filepath : string
+ __init__(filepath)
+ add(employee : AbstractEmployee) : int
+ get(id : int) : AbstractEmployee
+ get_all() : AbstractEmployee[]
+ get_all_by_type(type : string) : AbstractEmployee[]
+ update(employee : AbstractEmployee)
+ delete(id : int)
- _read_employees_from_file()
- _write_employees_to_file()
1
*
AbstractEmployee
- _id : int
- _first_name : string
- _last_name : string
- _employee_num : string
- _date_started : date
+ get_id() : int
+ set_id(id : int)
+ get_first_name() : string
+ get_last_name() : string
+ get_full_name() : string
+ get_employee_num() : string
+ get_years_employed() : float
+ to_dict() : dictionary
+ get_details() : string
+ get_type() : string
TeamManager
- _num_reports : int
- _team_name : string
+ get_num_reports() : int
+ get_team_name() :string
+ to_dict() : dictionary
+ get_details() : string
+ get_type() : string
Engineer
- _specialization : string
- _is_peng : boolean
+ get_specialization() : string
+ get_is_peng() : Boolean
+ to_dict() : dictionary
+ get_details() : string
+ get_type() : string

Design (4 marks)

UML: Update you UML in accordance with the highlighted changes above.

JSON: Design the JSON structure for each of your SpecificEntity classes. Create sample JSON for the following:

- SpecificEntity1 – Add
- SpecificEntity1 – Update
- SpecificEntity2 – Add
- SpecificEntity2 – Update

Use a type field in the JSON to distinguish each SpecificEntity so the API code can create the correct object (SpecificEntity1 or SpecificEntity2).

If you do not provide valid example JSON, you will lose 50% of your marks on the RESTful API component of this assignment because your marker will be unable to test it. Make sure they are valid and work with your RESTful API.

File Persistence (4 marks)

The Entity objects maintained in EntityManager will now persist to a text file in the JSON format you defined above. Each line in the file will be a string-based JSON representation of an Entity.

The following is a summary of the updated attributes and methods in your design related to file persistence:

- **_filepath** – The name and path of a text file where your Entity objects will be stored
- **_read_entities_from_file** – When you EntityManager is constructed (i.e., __init__), this method will load the JSON Entity records from the file at _filepath into your list of entities (i.e., _entities instance variable). Make sure it creates the correct type of Entity (SpecificEntity1 or SpecificEntity2).
- **_write_entities_to_file** – In any method that modifies an Entity object in your list of entities (i.e., _entities instance variable), this method is called. The first thing this method will do is open the file at _filepath for writing (such that the existing data will be overwritten). For each Entity record in _entities:
    - It will call the **to_dict()** method to get the Python dictionary will all the attributes for the SpecificEntity
    - It will serialize the Python dictionary to a JSON representation
    - The JSON representation is written as a string to a single line in the file

Your EntityManager should now be persisting the Entity objects from the _entities instance variable to the file at _filepath in a JSON format.

One further update: Modify your add method in EntityManager to return the Id of a successfully added Entity object.

Unit Testing (4 marks)

Update your unit tests as follows:

- SpecificEntity1 and SpecificEntity2 – Add a unit test to verify the new to_dict() method
- EntityManager – Modify your file as follows:
    - __init__ – Add tests for parameter validation on the filepath parameter
    - _read_entities_from_file – Mock this method so it does nothing. But make sure it is called upon construction of EntityManager.
    - _write_entities_from_file – Mock this method so it does nothing. But make sure it is called on each method in EntityManager that updates an Entity (i.e., add, update, delete).
    - add – test the newly returned Id value

RESTful API (15 marks)

Create a RESTful API for your EntityManager class with the following API endpoints. Update the naming of your API resources based on the names of your EntityManager and Entity classes (but don't use abstract in the entities resource name).

You must provide example JSON files (described in Design) for the add (POST) and update (PUT) API endpoints otherwise you will lose 50% of your marks on this section.

POST /entitymanager/entities

- Request: Entity JSON
- Response: Id of new Entity OR error message
- Returns 200 OK on success
- Returns 400 and an error message if the entity object is invalid

PUT /entitymanager/entities/<entity_id>

- Request: Entity JSON (without id)
- Response: None OR error message
- Returns 200 OK on success
- Returns 400 and an error message if the entity_id or entity object is invalid
- Returns 404 and an error message if the entity object does not exist

DELETE /entitymanager/entities/<entity_id>

- Request: None
- Response: None OR error message
- Returns 200 OK on success
- Returns 400 and an error message if the entity id is invalid
- Returns 404 and an error message if the entity object does not exist

GET /entitymanager/entities/<entity_id>

- Request: None
- Response: Entity JSON OR error message
- Returns 200 OK on success
- Returns 400 and an error message if the entity id is invalid
- Returns 404 and an error message if the entity object does not exist

GET /entitymanager/entities/all

- Request: None
- Response: List of Entity JSON (can be empty)
- Returns 200 OK on success

GET /entitymanager/entities/all/<type>

- Request: None
- Response: List of Entity JSON (can be empty) of the given type OR error message
- Returns 200 OK on success
- Returns 400 and an error message if the type is invalid

Make sure you test file persistence. If you shutdown or restart the Flask server, your Entity objects should be available when it is started again.

To ensure you thoroughly test your APIs, take two screenshots for each API endpoint:

- 1 success – 200 OK
- 1 error – 400 or 404

Make sure the Method, URI, Request, Response and Response Code is clearly visible in the screenshot. There should be a total or 12 screenshots that you will merge into a single file

Grading Summary

Design Updates • UML Updates (2 marks) • JSON Examples (2 marks)	4 marks
File Persistence Updates • Read on Construction (3 marks) • Write on Updates (3 marks) • JSON Storage (1 mark)	7 marks
Unit Test Updates • EntityManager test updates (3 marks) o Includes mocking • SpecificEntity test updates (1 mark)	4 marks
RESTful API • 6 Endpoints (12 marks – 2 marks each) • Test Screenshots (3 marks)	15 marks

No example JSON (from Design Updates) and you will lose 50% of your marks on this section.	
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 **Assignment2.zip** and upload to the dropbox in D2L (Activities -> Assignments -> Assignment 2):

- UML Design in PDF format (**uml_design.pdf**)
- Sample JSON files for Add (**add1.json and add2.json**)
- Sample JSON files for Update (**update1.json and update2.json**)
- Entity Manager Class (i.e., employer.py)
- Entity Manager Unit Test (i.e., test_employer.py)
- Abstract Entity Class (i.e., abstract_employee.py)
- Specific Entity 1 Class (i.e., manager.py)
- Specific Entity 1 Unit Test (i.e., test_manager.py)
- Specific Entity 2 Class (i.e., engineer.py)
- Specific Entity 2 Unit Test (i.e., test_engineer.py)
- RESTful API Module (i.e., employer_api.py)
- Screenshots – 12 in ONE PDF file (**screenshots.pdf**)

The submission is due Friday, March 22nd at midnight for all sets. No late submissions will be accepted.