Term Project: Iteration 2 – EventAttendee & EventManager

(Spring 2024)

Project Description:

In this iteration, we will create the two additional classes needed for the project: the EventAttendee class and the EventManager class.

The EventAttendee class is a simple class that acts as a tuple, attaching a specific Contact object to a specific Event object.

The EventManager class will manage the lists of events, contacts, event-attendee pairs, and other features necessary for managing the underlying classes.

All of the work that you do for this iteration will be found in the **classes** folder. Do not modify any other files, as that could easily lead to the program not running.

Instructions – EventAttendee Class:

In the file named EventAttendee.py, you will find an EventAttendee class. This is a simple class that acts as a tuple, attaching a specific Contact object to a specific Event object. This is used for the "Current Attendees" and "Add Attendees" features of the project. The class will have a `__str__` method already defined. Do not modify this method, unless you need to change attribute names for consistency.

In the EventAttendee class, add code to handle the following:

1. Create a constructor that accepts a Contact object and an Event object as arguments.
2. Save the Contact and Event objects as private attributes defined in the constructor. Additionally, create an attribute in the constructor called `self.__memo` and set it equal to an empty string.
3. Using the @property and @[object].setter decorators, create getters and setters for each of the three attributes defined in the previous step.

For now, this is all the code that is needed to make the EventAttendee class function. In future semesters, more features will be added to this class.

Instructions – EventManager:

In the file named EventManager.py, you will find an EventManager class. There is more work to be done in this class, as it will handle the lists of Contacts objects and Event objects for the entire project. The class will have two methods already defined:

- `def _sort_contacts(self)`
- `def _sort_events(self)`

These methods are used to sort the contact list and event list respectively. Do not modify these methods, unless you need to change attribute names for consistency.

In the EventManager class, add code to handle the following:

1. Create a constructor that accepts no arguments (apart from `self`) but initializes the following attributes:
    a. A list of Event objects, set as an empty list.
    b. A list of Contact objects, set as an empty list.
    c. A list of EventAttendee objects, set as an empty list.
    d. An Event UID, set to the integer 0. *(This will be incremented to determine the unique ID for each new event.)*
    e. A Contact UID, set to the integer 0. *(This will be incremented to determine the unique ID for each new contact.)*
2. Using the @property and @[object].setter decorators, create getters and setters for each of the five attributes defined in the previous step.
3. Define a method called **add_event** which accepts a dictionary object as an argument. This dictionary represents the new event to be added. This method should:
    a. Create a new Event object (using the dictionary object as input) and append it to the EventManager's list of events.
    b. Increment the value of the EventManager's Event UID attribute, so that the next event added will have a different unique ID.
    c. Sort the EventManager's list of events by calling the pre-existing `_sort_events` method.
4. Define a method called **add_contact** which accepts a dictionary object as an argument. This dictionary represents the new contact to be added. This method should:
    d. Create a new Contact object (using the dictionary object as input) and append it to the EventManager's list of contacts.
    e. Increment the value of the EventManager's Contact UID attribute, so that the next contact added will have a different unique ID.
    f. Sort the EventManager's list of contacts by calling the pre-existing `_sort_contacts` method.
5. Define a method called **is_attending** which accepts a Contact object and an Event object as arguments. This method should:

    a. Iterate through the EventManager's list of EventAttendees to determine whether the given contact argument is attending the given event argument.

    b. If the contact is attending the event, the method should return True.

    c. If the contact is not attending the event, the method should return False.

6. Define a method called **add_event_attendee** which accepts a Contact object and an Event object as arguments. These are the contact and event that will be paired together in the new EventAttendee object. This method should:

    a. Check to see whether the given contact is already attending the given event.

    b. If the contact is not already attending the event, create a new EventAttendee object (using the Contact and Event objects as input) and append it to the EventManager's list of EventAttendees.

    c. If the contact is already attending the event, do nothing.

7. Define a method called **uid_to_event** which accepts an integer (representing a unique ID) as an argument. This method should:

    a. Iterate through the EventManager's list of events to determine whether any event has the given integer as an unique ID.

    b. If a matching ID is found, the method should return the Event object associated with it.

    c. If no matching ID is found, the method should return None.

8. Define a method called **uid_to_contact** which accepts an integer (representing a unique ID) as an argument. This method should:

    a. Iterate through the EventManager's list of contacts to determine whether any contact has the given integer as an unique ID.

    b. If a matching ID is found, the method should return the Contact object associated with it.

    c. If no matching ID is found, the method should return None.

Instructions – Running the Program:

Once you have completed the above steps, your program should be ready to run. To run the program, do the following:

1. Install the **python-Levenshtein** package. This package is necessary for the project's search function to work.

    a. In PyCharm, navigate to the Packages tab. This will be at the bottom of your PyCharm, where the console is displayed. There is a button on the left side of the console that resembles three tiles stacked atop one another.

   b. Search for "python-Levenshtein**"** in the search bar of the packages tab. Once you've found it, click the Install package button on the right of the console area.

   c. After a few moments, the package should be installed. You may see a green loading bar in your PyCharm window.

2. If you have not done so already, add your Contact.py and Event.py files from Iteration 1 to the project's **classes** folder. If you completed Iteration 1 correctly, your classes will be compatible with the other files. If you did not complete Iteration 1 correctly, you may need to debug your Contact.py and Event.py files.

3. In PyCharm, open the main.py file associated with this project. Once the file is opened, click the Run button as you would for any other program.

4. If you have completed the Iteration correctly, you will see the GUI appear and you will have access to all the features shown in class, such as Display Contacts, Display Events, Add Contact, Add Event, and the other functions associated with them.

5. When moving projects from one workstation to another, errors can always occur. If you are unable to run the program, but you are confident that the issue is not related to the code that you have written, please reach out to one of the TAs. They should be able to help you troubleshoot the issue.

Deliverables:

Place all files needed to run the program (including your Contact.py and Event.py classes) in a single .zip file named userid_iteration_2.zip, where "userid" is your Tech username. For example, "jstrickler_iteration_2.zip." Submit the .zip file to the Iteration 2 dropbox in iLearn.

NOTE: Whether or not your program runs (and does so correctly) will be the biggest determining factor for your grade in this iteration. Make sure your program runs on your machine before you submit it. If you want, I do not mind you submitting a video of yourself running the program on your machine (along with your other files, of course), just in case something bizarre happens with your submission. You do not have to do this; it's just an option available to you.