COMP1112 Object Oriented Programing
Spring 2020, Project#1
Due Date: 26 March, 2020 23:50

IŞIK UNIVERSITY
COMPUTER SCIENCE AND ENGINEERING

Design Submission: You should upload the UML diagram to the system https://isikuniversity.mrooms.net/ as Visual Paradigm file with vpp extension. Please **DO NOT submit by e-mail**. Everyone will get feedback about design on.

Project Submission: Upload corrected UML diagram and compressed project folder through the system https://isikuniversity.mrooms.net/ . Please DO NOT submit by e-mail.

If you have any questions, please either send e-mail to ilknur.karadeniz@isikun.edu.tr or visit me in my office.

You are supposed to model and implement a personal appointment/meeting book to save a person's meeting schedule.

- `Person` class:
  - Attributes:
    - `name`: set during construction, `id:` must be unique
    - `myMeetings`: the list of meetings of the person that s/he plans to attend
    - `iOrganize`: the list of meetings organized by a person
  - Methods:
    - Constructor(s): requires `name` to be constructed.
    - Required accessor/mutators
    - `equals()`: override this method to check whether two `Person` instances are the same, i.e., name and id fields must be the same.
    - `addMeeting` ( ): takes a `Meeting` instance, adds to the `myMeetings` list, only if the person doesn't have another meeting at the specified meeting time. If the meeting instance is successfully added to the list, returns `true`.
    - `removeMeeting` (): takes a `Meeting` instance, removes the instance from `myMeetings` list.
    - `organizeMeeting`(): takes a `Meeting` instance, adds the instance to the `iOrganize` list.
    - `cancelMeeting`(): used to cancel the Meeting instance which is taken as argument, organized by the Person under inspection. The meeting to be cancelled should be removed from all its participants' lists.
    - `displayMyMeetings`(): displays the whole list of meetings' dates, the host of the meeting.
    - `displayMyOrgnizations`(): displays the Meetings organized by the Person under inspection.
    - `toString()`: returns important info of `Person` instance as `String`.

`Meeting` class:
  - Attributes:
    - `date`: the date/time of appointment, can be changed only to a later date.
    - `attendees:` list of `Person` instances invited to the `Meeting` instance.
    - `host`: a `Person` instance, the owner of the `Meeting` instance, must be set by the constructor, cannot be changed.
  - Methods
    - Constructor(s); requires the date, the host, and **at least one attendee** to construct the `Meeting` instance. The current `Meeting` instance should be added to the host's, and the attendees lists as well.

Simplicity is prerequisite for reliability.
--Edsger W.Dijkstra

- Required accessor/mutators
- `equals()`: in order for the two meeting instances to be equal, the date fields, the hosts, and the list of attendees must be the same.
- `addAttendee`(): takes a `Person` instance, if the person is not in the list, then the person's `addMeeting`() method is invoked. If `true` is returned, then the person is added to `attendees`.
- `removeAttendee`(): takes a `Person` instance. If he is in the `attendees`, removes him, invokes the person's `removeMeeting()`, and returns `true`.
- `removeAllAttendees`(): removes all attendees of the event. Required if meeting needs to be cancelled.
- `toString()`: returns host, date, and list of attendees as String instance.

a. **Draw the UML diagram for the classes. Make sure you show modifiers.**
b. Implement the classes.
c. Write a test class. This test program will first display a main menu like the following (an output file has been attached to this document):
  - **a.** Create and host a new meeting (you must know at least one friend who will attend your event)
  - **b.** Cancel a meeting:
  - **c.** Attend an existing meeting
  - **d.** Leave a meeting
  - **e.** Display my Meetings
  - **f.** Display Meetings organized by me
  - **g.** Logout
  - **h.** Exit: quits the app.

Each option runs as follows:

a. Create and host a new meeting (you must know at least one friend who will attend your event): `createMeeting`() is invoked. It asks the user the date, and the name of the meeting. The user who has logged in will be the host of the meeting. Then, the `Meeting` instance is created, added to the current user's `iOrganize` list by invoking `organizeMeeting`(), and added to the `allMeetings` list of `TestClass`.
Returns to the main menu.
b. Cancel a meeting: `cancelMeeting`() is invoked. Only hosts are allowed to cancel meetings, which are created by them. So, all meetings hosted by the current user are displayed(menu item f), and asked the name of which is to be cancelled. The meeting, say `cancelMe`, is fetched from `allMeetings` list of `TestClass`. Current user's (supposed to be the host) `cancelMeeting`() method is invoked by passing `cancelMe` meeting instance as an argument.
Returns to the main menu.
c. Attend an existing meeting: `attendMeeting` () will be invoked. First the list of meetings is displayed. Then the user is asked if s/he would like to attend any of them. If so, s/he is asked which meeting to attend, and s/he is added to the attendee list of the chosen meeting.
d. Leave a meeting: `leaveMeeting` () will be invoked all meetings of the user is displayed. Then the user is asked which meeting is to leave.
e. Display my Meetings: displays all meetings currents user is attending.
f. Display Meetings organized by me: displays all the meetings organized by the current user.
g. logout: current user is logged out, and returns to the main menu.