Homework assignment 2.
Abstract programming

1 Introduction

The goal of this assignment is to practice skills related to the following topics:

- Mutable and immutable objects
- Delegates and events
- Interfaces
- Inheritance

The assignment should be completed using the supplied Visual Studio template project (PersonalScheduler). It is a WPF application, however only WPF basics are required for completion of the assignment. Most of the parts related to GUI are already implemented, you will need to make a couple of adjustments. Feel free to post questions on the Canvas forum about WPF features.

Application idea

The application allows to manage important events and to make notifications about them to the user as their time is reached in several different ways - visual, sound and email.

First build your solution, it will restore some external packages, then run the application. Notice that the program minimizes to tray (panel in the bottom right corner of the screen) when clicking the close button. To close it completely open the context menu of the tray icon by right-clicking it and select Exit.

Take some time to study the given project code. Below are brief descriptions of source files:

- **MainWindow** - main window of the application, displays a list of coming events and has two buttons - add and remove. Their click handlers are already implemented.

  The MainWindow class (check MainWindow.xaml.cs) contains a timer, which fires events each second, the event handler calls the "ProcessEvents" method of the EventManager class (which you will need to implement)

- **EventInfoWindow** - helper window used to fill in information about a new event
- **EventManager** - repository-like class used to store information about all events. The main logic for issuing notifications will reside in the "ProcessEvents" method
- **ScheduledEvent** - the main model class (should contain data related to a single event)
- **VisualNotifier** - a notifier class responsible for showing visual messages - the current implementation is a temporary one, you will need to change it
- **SoundNotifier** - a notifier class responsible for playing a standard system sound, the implementation can be kept as it is.

Two of the window classes contain a "TemplateCode" region - you can unfold it to look through the implementation details but you don't need to change them

2 Description of tasks

Corrections are shown in purple
Self-check steps are shown in blue

1. (1.5 points) Add the following properties to the "ScheduledEvent" class:
   - Name (string)
   - DateTime (DateTime)
   - Description (string)
   - Place (string)
   - Notifications (List of NotificationType)

   Design the ScheduledEvent class in such a way that its objects are immutable

   The following restrictions on ScheduledEvent data are imposed:

   - Name cannot be set as null (throw ArgumentNullException), an empty string or a string of whitespaces (throw ArgumentException)
   - DateTime cannot be earlier than the current moment (the moment when a new ScheduledEvent object is created) - throw ArgumentOutOfRangeException
   - At least one notification has to be specified for an event (otherwise throw an ArgumentException)

   A class should contain a single constructor, accepting all parameters in the same order as the properties are listed above.

2. Navigate to the "EventManager" class, which is designed to store and manage a list of events. Add a private field of the standard List type to store all scheduled events. **The list should always be sorted by event datetime in ascending order.**

3. (1.5 points) Implement the AddEvent method of the EventManager class. It should insert the new event to an appropriate position in the event list, so that the sorted order is kept. In addition the method should check for event duplicates (by name), in case a matching event is found, an ArgumentException should be raised with an appropriate message. Exception handling should be performed in UI-related classes.

   Implement the RemoveEvent method.

4. (1 point) Declare two events (C# events) inside the EventManager class: one will be used when adding a new scheduled event, the other - when removing an existing event. Events should be handled inside the MainWindow class to update the event listbox (check a similar Repository example from week 3)

   Complete logic of the buttonOK_Click handler inside EventInfoWindow: using data from UI controls it should create a new event and add it to the EventManager instance.

   At this stage the user should be able to enter information about a new event and see its name and date appear in the listbox as well as to remove an existing event by selecting an item in the listbox and pressing "Delete".

5. (2.5 points) It is now time to implement the central part of the application, i.e. the notification logic. Study the two provided notifier classes and declare a common interface for them - this is the only time when you might need to change method signatures.

   Change implementation of the VisualNotifier. Instead of calling a standard MessageBox (which blocks the whole application) the VisualNotifier should show a new WPF window (you need to design it yourself), in which information about the new event will be displayed. Implementation of the SoundNotifier can be kept as it is.

   Implement the ProcessEvents method inside the EventManager class. As mentioned before, this method is called regularly (each second) from the timer handler and its task it to check whether event time has approached. When the current date and time become equal or greater than those of the some event in the list, notifications for that event have to be made through all requested channels (visual, sound or email: each event has its own set of channels). After issuing all notifications requested by the user, an event can be deleted from the list (or rescheduled in case of a regular event - see next task).

6. (2.5 points) Add a subclass of the ScheduledEvent called RegularEvent. It should have an additional RepeatInterval property of type TimeSpan. When inside the EventInfoWindow the repeat checkbox is activated a new event should be created as a regular event. Additional validation should be performed when creating a RegularEvent instance:

   - Email notifications cannot be used for regular events with repeat interval less than 5 minutes (as they may be treated as spam) - throw an ArgumentException inside the RegularEvent class constructor.

   When a regular event is fired, apart from issuing all necessary notifications the event should be rescheduled for a later time by adding the repeat interval to the previous date (e.g. when the initial date was set to Oct 3rd 2016 22:00 and repeat interval is 10 minutes, the next notification sequence should occur at 22:10 of the same day)

7. (1 point) Check online resources explaining how a C# application can send emails[1]. Add a new EmailNotifier class to the project. The target email address, to which email notifications will be sent, should be passed in the class constructor. **Before submitting your solution make sure you delete all personal data from the application code, e.g. email account username and password**

Extra requirements and recommendations:

- In the final implementation the EventManager should not have direct references to either the Notifier classes or the RegularEvent class derived from ScheduledEvent. The exact way of how you remove these references does not matter, you can use any of the techniques we studied
- Apart from a single exception (see p. 5 of tasks), don't change signatures (names, return types, parameters) of existing methods
- All exceptions that you add to the model / logic classes as well as possible standard exceptions (e.g. converting from string to int) have to be handled inside the UI. The application should not crash because of the user entering incorrect data.

3 Submission

Submit your solution following these steps:

1. Delete three temporary subfolders - "bin", "obj" from the project folder and "packages" (**not the packages.config file**) from the solution folder.

[1] If you plan to use the gmail server, verify that you are using secure connection (port 587). If you have two-step verification enabled for your gmail account you might need to generate an application-specific password - see related resources

2. Add the whole solution folder to a ZIP **(not RAR or 7Z)** archive.

3. Upload the archive to the Canvas LMS in the corresponding section

4 Grading policy

The final grade for the assignment is determined by the number and quality of completed tasks. An instructor has the option to ask one or two additional questions in case of an unclear grade.

The grade can be lowered in the following cases:

- Inefficient implementation of an algorithm (-1 point)
- Poor programming style (-1 point) (ask your instructor for the definition of "poor")