**Государственное бюджетное учреждение г. Москвы**

**Лицей №1586**

**Проектная работа**

**Программа, которая почему-то решает СЛАУ**

**Подготовлен**

**Скоробогатов Денис**

**Проверяет**

**Луговской Кирилл Игоревич**

# Введение

«Сохраняйте, организуйте и делитесь заметками где угодно. Любые ваши идеи будут всегда под рукой, синхронизированы и доступны для работы.»

Команда EverNote.

**Органа́йзер** (англ. *organizer*) — изначально небольшая книга, содержащая календарь, адресную книгу и блокнот, служащая для организации информации о личных контактах и событиях. С развитием информационных технологий книга стала заменяться сначала электронными органайзерами, затем карманными персональными компьютерами, компьютерными программами и онлайн-органайзерами, обладающими дополнительными функциями: напоминание о предстоящих событиях, защита и синхронизация информации.

Органайзер является средством управления временем. Предварительное планирование дел помогает повысить плодотворность любой деятельности, как личной, так и профессиональной. Поэтому и человека, который консультирует организации и частных лиц в сфере повышения эффективности управления временем, тоже иногда называют «органайзер»; в США существует Национальная ассоциация профессиональных органайзеров (NAPO).

В XXI веке под органайзером чаще понимается программное обеспечение для ПК и мобильных устройств.

Оглавление

[Введение 2](#_Toc484672812)

[Постановка задачи 4](#_Toc484672813)

[Описание теоретической части. 5](#_Toc484672814)

[Описание практической части 6](#_Toc484672815)

[Заключение 12](#_Toc484672816)

[Список литературы 13](#_Toc484672817)

[Приложение 1. 14](#_Toc484672818)

# Постановка задачи

Задача проста: создать простое приложение для выбранной платформы, способное выполнять базовый функционал органайзера с использованием возможностей сетевого сервиса “Evernote” (<evernote.com>). Необходимый функционал:

* Создание в приложении заметок, содержащий
  + Название
  + Описание
  + Дату
  + Адрес
  + Дату уведомления о событии
  + Текст уведомления о событии
* Создание приложения, способного работать с заметками, а именно:
  + Создавать заметки
  + Удалять заметки
  + Выдавать напоминания в момент наступления даты уведомления о событии
* Синхронизация заметок в приложении с облаком с помощью *Evernote*

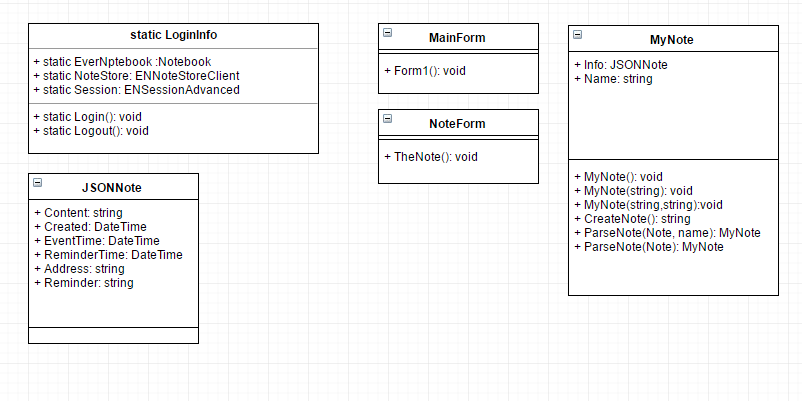
# Описание теоретической части.

1. Выбор платформы. На данный момент существуют пять самых популярных платформ для разработки: *ОС* *Windows, ОС Linux (Ubuntu, Debian, Fedora и т.д.), Mac OS X,* *Android* и *iOS*. В результате, была выбрана платформа *Windows*, так как разработка под эту платформу имеет следующие преимущества:
   1. Удобство разработки на нативном .*Net*, используя в качестве среды знакомую *Microsoft Visual Studio 2015* Community, и знакомый язык – *C#*;
   2. Документация по *Evernote* *SDK* (<dev.evernote.com/doc/>) – несмотря на то, что документация доступна для 12 различных языков, документация по Windows (C*#),* *PHP* и *iOS* (*Objective-C*) наиболее полная – несмотря даже на то, что является не полностью достоверной.
   3. Отсутствие нужды использовать напрямую *фреймворк* *Apache* *Thrift*, на котором построен *Evernote* *API*, а так же наличие встроенного модуля *OAuth* для авторизации (*OAuth* является единственный валидным способом авторизоваться на сервере *Evernote*, но не во всех *Evernote* *SDK* он имеется).
   4. Простая установка *EvernoteSDK* через *NuGet* в *MVC*.
2. *Evernote* *API*. У этого *API* есть много вариантов *SDK*, как говорилось ранее. У *SDK* для *ОС* *Windows* есть три «уровня» *SDK*:
   1. *EvernoteSDK* – базовый уровень, просто в использовании, но в нем слишком мало методов и классов для полноценной разработки – ([github](https://github.com/evernote/evernote-cloud-sdk-windows/blob/master/Getting_Started.md)).
   2. *EvernoteSDK.Advanced* – «продвинутый уровень», дающий большую свободу действий, но доступ к методам, классам и типам, описанным в [Evernote API](http://dev.evernote.com/doc/reference/), все еще невозможен.
   3. *Evernote.EDAM* – *SDK*, обеспечивающий поддержку *EvernoteAPI*, но реализация методов в этом *SDK* несколько отличается от документации *API*, что затрудняет процесс подготовки программы.
3. Дополнительные сторонние библиотеки/*SDK*:
   1. *AngleSharp* – зависимость *PreMailer*.
   2. *PreMailer* – зависимость *Evernote* *SDK*
   3. *Newtonsoft.JSON* – необходимость (де)сериализации *JSON*
   4. *CsQuery* – зависимость *Evernote* *SDK*
   5. *Apache* *Thrift* – зависимость *Evernote* *SDK*
   6. *SgmlReader* – зависимость *Evernote* *SDK*

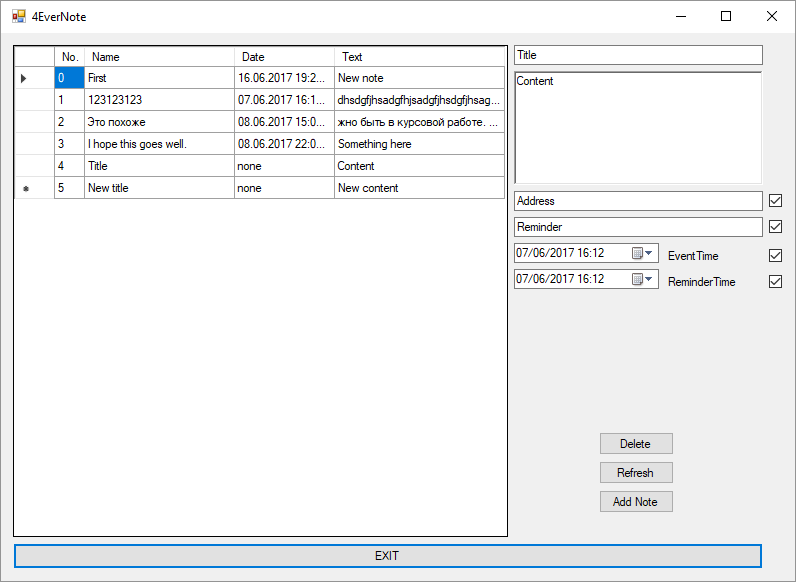
# Описание практической части

Для реализации была выбрана платформа *Windows* *10*, среда разработки *Microsoft* *Visual* *Stuido*, язык *C#* и *Evernote* *SDK* для *Windows*. Код программы находится в Приложениях к этой работе.

Диаграмма классов:



MainForm



Как видно, именно это класс отвечает за *GUI* и, фактически, связывает все компоненты программы. Таблица заметок представлена с помощью *DataGridView*, справа находятся поля для создания новой заметки – *textbox*, *richTextBox* и *DateTimePicker* – последние четыре, отвечающие, соответственно, за адрес, текст напоминания, дату события и дату напоминания, не являются обязательными, что является интуитивно понятным, так как слева расположены элементы *CheckBox*, которые можно отметить, таким образом, включив или отключив данный компонент создаваемой заметки.

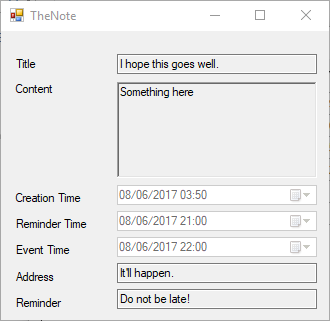
Кнопки *Delete*, *Refresh* и *Add* *Note* отвечают за:

* *Delete* – удаление заметки. При этом, для визуального получения результата удаления, необходимо нажать на Refresh, обновив таблицу слева.
* *Refresh* – обновление таблицы – по нажатию данной кнопки программа получает с сервера актуальный список заметок и соответственно обновляет таблицу.
* *Add* *Note* – добавляет новую заметку и отправляет ее на сервер. Данные заметки заполняются выше.

Кнопка *EXIT* отвечает за *Logout* – выход из аккаунта. По простому закрытию приложения данные пользователя сохраняются.

При двойном клике на любую из строк таблицы, на экран выводится следующий элемент GUI – *NoteForm*.

NoteForm



Это *Read-Only* форма, имеющая несколько полей, каждое из которых содержит информацию о заметке. Эти поля могут быть как пустыми, так и полными. Ограничения на символы нет.

LoginInfo

Данный класс отвечает за процедуру авторизации и выхода. Он имеет методы *void* *Login()* и *void* *Logout(),* которые, очевидно, позволяют пользователю авторизоваться и выйти из аккаунта. Этот класс является статическим, и используется при каждом запуске программы. Использует технологию OAuth для авторизации. У этого класса есть несколько public значений:

* *<Notebook> EverNotebook* – этот объект содержит в себе информацию о «блокноте» внутри аккаунта пользователя, в который *4EverNote* ведет запись
* *<ENNoteStoreClient>* *NoteStore* – информация о неком сервисе, который используется для доступа к *Evernote.EDAM* *SDK*
* *<ENSessionAdvanced> Session* – информация о текущей сессии.

MyNote

Данный класс был реализован в замену *ENNote*, т.к. последний имел скудные возможности, плохую документацию и ограниченные возможности. Идея в том, что *ENNote*.Content присваивается строка с сериализованной JSON-информацией, с которой уже работает класс *MyNote* и *4EverNote*. Данный класс имеет следующие публичные поля:

* *<JSONNote> Info* – объект, содержащий основную смысловую информацию заметки
* *<string> Name* – имя заметки

Публичные методы класса:

* *<string> CreateNote()* – сериализует информацию и создает заметку с помощью приватного метода *FormatNote()*, а так же загружает ее на сервер. Возвращает имя загруженной заметки.
* *static <MyNote> ParseNote(Note) + 1* – имеет перегрузку *(Note, string)* – в качестве Note получает скачанную с сервера заметку, а в качестве name – имя этой заметки (необязательно). После этого данный метод формирует сначала объект типа *JSONNote*, получая информацию, десериализуюя *JSON*-сериализованный текст в *Note.Content*, и возвращает готовую заметку типа *MyNote*.

JSONNote

Используется классом *MyNote*, и имеет следующие публичные поля:

* *<string> Content* – текст заметки
* *<DateTime> Created* – дата создания
* *<DateTime> EventTime* – дата события
* *<DateTime> ReminerTime* – дата напоминания о событии
* *<string> Address* – местонахождение события
* *<string> Reminder* – текст напоминания

# Заключение

Метод Гаусса является мощным и простым для понимания математическим инструментом для решения СЛАУ; его недостатки компенсируются простотой алгоритма; время, затраченное на подготовку проекта, отлично заменило мне сон.

# Список литературы

* <https://evernote.com/>
* <http://dev.evernote.com/>
* <https://stackoverflow.com/>
* <https://discussion.evernote.com/>
* <https://msdn.microsoft.com/>
* <https://sandbox.evernote.com/>
* <https://github.com/>
* <https://www.draw.io/>
* <https://docs.microsoft.com/>
* <http://www.cyberforum.ru>

# Приложение 1.

Исходный код MainForm (Form1.cs)

**using** **System**;

**using** **System.Windows.Forms**;

**using** **Evernote.EDAM.NoteStore**;

**using** **Evernote.EDAM.Type**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Media**;

**namespace** **\_4EverNote**

{

**public** **partial** **class** **Form1** : Form

{

**private** Dictionary<**int**, **string**> id\_guid;

**private** SortedDictionary<DateTime, **string**> date\_guid;

**private** ICollection<DateTime> keys;

**public** **Form1**()

{

InitializeComponent();

id\_guid = **new** Dictionary<**int**, **string**>();

date\_guid = **new** SortedDictionary<DateTime, **string**>();

}

**private** **void** **Form1\_Load**(**object** sender, EventArgs e)

{

**try**

{

// logging in

LoginInfo.Login();

} **catch** (NullReferenceException)

{

// that happens if the login window is closed before auth

Application.Exit();

}

RefreshGrid();

}

/// <summary>

/// Refreshes the table

/// </summary>

**private** **void** **RefreshGrid**()

{

// clear all the tables and rows

dataGrid.Rows.Clear();

id\_guid.Clear();

date\_guid.Clear();

**int** i = **0**;

**int** j = **0**;

**bool** invalid = **false**;

// download all the notes from user's account, form them into list

**var** a = **new** NoteFilter();

a.NotebookGuid = LoginInfo.EverNotebook.Guid;

NoteList nlist = LoginInfo.NoteStore.FindNotes(a, **0**, **1000**);

List<Note> \_nlist = nlist.Notes;

**if** (\_nlist.Count == dataGrid.Rows.Count - **1**)

**return**;

// let the magic happen

**foreach** (**var** note **in** \_nlist)

{

// parse each note into MyNote

MyNote parsednote;

parsednote = MyNote.ParseNote(LoginInfo.NoteStore.GetNote(note.Guid, **true**, **false**, **false**, **false**), note.Title);

**if** (parsednote == **null**)

**continue**;

// add a new row

**if** (\_nlist.Count > dataGrid.Rows.Count)

dataGrid.Rows.Add();

// place some data into the table

dataGrid.Rows[i].Cells[**0**].Value = i;

id\_guid.Add(i, note.Guid);

**while** (invalid == **false**)

{

**try**

{

**if** (DateTime.Compare(DateTime.Now, parsednote.Info.ReminderTime) < **0**)

{

date\_guid.Add(parsednote.Info.ReminderTime.AddSeconds(j), note.Guid);

}

invalid = **true**;

}

**catch** (ArgumentException)

{

j++;

}

}

dataGrid.Rows[i].Cells[**1**].Value = parsednote.Name;

**if** (parsednote.Info.EventTime != **null**)

{

**if** (parsednote.Info.EventTime == DateTime.MinValue.AddYears(**1755**))

dataGrid.Rows[i].Cells[**2**].Value = "none";

**else**

dataGrid.Rows[i].Cells[**2**].Value = parsednote.Info.EventTime;

}

**if** (parsednote.Info.Content != **null**)

{

**if** (parsednote.Info.Content.Length < **15**)

dataGrid.Rows[i].Cells[**3**].Value = parsednote.Info.Content;

**else**

dataGrid.Rows[i].Cells[**3**].Value = parsednote.Info.Content.Remove(**0**, **14**) + "...";

}

i++;

invalid = **false**;

}

// "sorting" notes using reminder's time

keys = date\_guid.Keys;

}

**private** **void** **exit\_Click**(**object** sender, EventArgs e)

{

// logging out

LoginInfo.Logout();

Application.Exit();

}

**private** **void** **addNote\_Click**(**object** sender, EventArgs e)

{

**var** a = **new** MyNote(titlebox.Text, contentbox.Text);

// address

**if** (isAddress.Checked == **true**)

a.Info.Address = addressbox.Text;

**else**

a.Info.Address = **null**;

// reminder text

**if** (isReminder.Checked == **true**)

a.Info.Reminder = reminderbox.Text;

**else**

a.Info.Reminder = **null**;

a.Info.Created = DateTime.Now;

// event time

**if** (isEventTime.Checked == **true**)

a.Info.EventTime = eventTime.Value;

**else**

a.Info.EventTime = DateTime.MinValue.AddYears(**1755**);

// reminder time

**if** (isReminderTime.Checked == **true**)

a.Info.ReminderTime = reminderTime.Value;

**else**

a.Info.ReminderTime = DateTime.MinValue.AddYears(**1755**);

// creating a note and refreshing the grid

a.CreateNote();

RefreshGrid();

}

**private** **void** **refresh\_Click**(**object** sender, EventArgs e)

{

RefreshGrid();

}

**private** **void** **dataGrid\_CellDoubleClick**(**object** sender, DataGridViewCellEventArgs e)

{

**var** thisRow = dataGrid.CurrentRow;

**string** guid = id\_guid[Int32.Parse(thisRow.Cells[**0**].Value.ToString())];

**var** window = **new** TheNote(MyNote.ParseNote(LoginInfo.NoteStore.GetNote(guid, **true**, **false**, **false**, **false**)), guid);

window.Show();

}

**private** **void** **delete\_Click**(**object** sender, EventArgs e)

{

**var** thisRow = dataGrid.CurrentRow;

**string** guid = id\_guid[Int32.Parse(thisRow.Cells[**0**].Value.ToString())];

LoginInfo.NoteStore.DeleteNote(guid);

}

**private** **void** **isAddress\_CheckedChanged**(**object** sender, EventArgs e)

{

addressbox.ReadOnly = **true**;

}

**private** **void** **isReminder\_CheckedChanged**(**object** sender, EventArgs e)

{

reminderbox.ReadOnly = **true**;

}

**private** **void** **isEventTime\_CheckedChanged**(**object** sender, EventArgs e)

{

eventTime.Enabled = **false**;

}

**private** **void** **isReminderTime\_CheckedChanged**(**object** sender, EventArgs e)

{

reminderTime.Enabled = **false**;

}

// timer for reminders

**private** **void** **timer1\_Tick**(**object** sender, EventArgs e)

{

// if (keys.First() < DateTime.Now.AddMinutes(1) && keys.First() > DateTime.Now)

// if the time is right - show the event info

**if** (DateTime.Compare(keys.First(), DateTime.Now.AddMinutes(**1**)) <= **0** && DateTime.Compare(keys.First(), DateTime.Now) >= **0**)

{

**string** guid = date\_guid[keys.First()];

date\_guid.Remove(keys.First());

**var** window = **new** TheNote(MyNote.ParseNote(LoginInfo.NoteStore.GetNote(guid, **true**, **false**, **false**, **false**)), guid);

SystemSounds.Beep.Play(); // dat sound

window.TopMost = **true**;

window.Show();

}

}

}

}

Код NoteForm (TheNote.cs)

**using** **System**;

**using** **System.Windows.Forms**;

**namespace** **\_4EverNote**

{

**public** **partial** **class** **TheNote** : Form

{

**private** MyNote \_note;

**private** **string** \_guid;

**public** **TheNote**(MyNote note, **string** guid)

{

InitializeComponent();

\_note = note;

\_guid = guid;

}

**private** **void** **TheNote\_Load**(**object** sender, EventArgs e)

{

title.Text = \_note.Name;

content.Text = \_note.Info.Content;

creation.Value = \_note.Info.Created;

eventTime.Value = \_note.Info.EventTime;

reminder.Value = \_note.Info.ReminderTime;

reminderText.Text = \_note.Info.Reminder;

address.Text = \_note.Info.Address;

**if** (eventTime.Value == DateTime.MinValue.AddYears(**1755**)) // if the date is not set - let's set it to a minimum possible date! (almost)

eventTime.Visible = **false**;

**if** (reminder.Value == DateTime.MinValue.AddYears(**1755**))

reminder.Visible = **false**;

}

}

}

Код LoginInfo (LoginInfo.cs)

**using** **Evernote.EDAM.Type**;

**using** **EvernoteSDK.Advanced**;

**namespace** **\_4EverNote**

{

**static** **class** **LoginInfo**

{

**static** **private** Notebook \_noteBook;

**static** **private** ENNoteStoreClient \_noteStore;

**static** **private** ENSessionAdvanced \_session;

**static** **public** Notebook EverNotebook

{

**get** { **return** \_noteBook; }

}

**static** **public** ENNoteStoreClient NoteStore

{

**get** { **return** \_noteStore; }

}

**static** **public** ENSessionAdvanced Session

{

**get** { **return** \_session; }

}

**static** **public** **void** **Login**()

{

**var** notebookExists = **false**;

// log in to Evernote cloud service

ENSessionAdvanced.SetSharedSessionConsumerKey("endlessborland-5836", "e83cae88abd45ca7", "sandbox.evernote.com");

**if** (ENSessionAdvanced.SharedSession.IsAuthenticated == **false**)

{

ENSessionAdvanced.SharedSession.AuthenticateToEvernote();

}

\_session = ENSessionAdvanced.SharedSession;

// creating a NoteStore for advanced functionality

\_noteStore = ENSessionAdvanced.SharedSession.PrimaryNoteStore;

**var** NotebookList = \_noteStore.ListNotebooks();

**foreach** (**var** a **in** NotebookList)

{

**if** (a.Name == "4EverNote")

{

notebookExists = **true**;

\_noteBook = a;

}

}

**if** (notebookExists == **false**)

{

**var** notebook = **new** Notebook();

notebook.Name = "4EverNote";

\_noteBook = \_noteStore.CreateNotebook(notebook);

}

NotebookList.Clear();

NotebookList = **null**;

}

**static** **public** **void** **Logout**()

{

ENSessionAdvanced.SharedSession.Preferences.RemoveAllObjects();

}

}

}

Код MyNote (MyNote.cs)

**using** **Evernote.EDAM.Type**;

**using** **EvernoteSDK.Advanced**;

**using** **Newtonsoft.Json**;

**using** **System.Text.RegularExpressions**;

**namespace** **\_4EverNote**

{

**public** **class** **MyNote**

{

**private** **string** \_name;

**private** Note newNote;

/// <summary>

/// All the info inside the ENNote.Content

/// </summary>

**public** JSONNote Info;

**public** **string** Name

{

**get**

{

**return** \_name;

}

**set**

{

\_name = **value**;

}

}

**public** **MyNote**()

{

newNote = **new** Note();

Info = **new** JSONNote();

}

**public** **MyNote**(**string** name)

{

newNote = **new** Note();

Info = **new** JSONNote();

Name = name;

}

**public** **MyNote**(**string** name, **string** content)

{

newNote = **new** Note();

Info = **new** JSONNote();

Name = name;

Info.Content = content;

}

/// <summary>

/// Makes a string of JSON

/// </summary>

**private** **void** **FormatNote**()

{

**string** serizlized = JsonConvert.SerializeObject(Info);

newNote.Title = Name.ToString();

**var** writer = **new** ENMLWriter();

writer.WriteStartDocument();

writer.WriteString("4evernote-5836");

writer.WriteString(serizlized);

writer.WriteEndDocument();

newNote.Content = writer.Contents.ToString();

newNote.NotebookGuid = LoginInfo.EverNotebook.Guid;

}

/// <summary>

/// Uploading the note on the server

/// </summary>

/// <returns>Note title</returns>

**public** **string** **CreateNote**()

{

FormatNote();

**var** result = LoginInfo.NoteStore.CreateNote(newNote);

**return** result.Title;

}

/// <summary>

/// Parses the JSON into MyNote

/// </summary>

/// <param name="one">Downloaded Evernote.EDAM.Type.Note</param>

/// <param name="name">Note name</param>

/// <returns></returns>

**public** **static** MyNote **ParseNote**(Note one, **string** name)

{

MyNote n = **new** MyNote();

**string** con = Regex.Match(one.Content, @"(?<=<en-note>)(.\*)(?=</en-note>)").ToString();

**if** (con.StartsWith("4evernote-5836") == **false**)

**return** **null**;

con = con.Substring(**14**);

n.Info = JsonConvert.DeserializeObject<JSONNote>(con);

n.Name = name;

**return** n;

}

/// <summary>

/// Parses the JSON into MyNote

/// </summary>

/// <param name="one">Downloaded Evernote.EDAM.Type.Note</param>

/// <returns></returns>

**public** **static** MyNote **ParseNote**(Note one)

{

MyNote n = **new** MyNote();

**string** con = Regex.Match(one.Content, @"(?<=<en-note>)(.\*)(?=</en-note>)").ToString();

**if** (con.StartsWith("4evernote-5836") == **false**)

**return** **null**;

con = con.Substring(**14**);

n.Info = JsonConvert.DeserializeObject<JSONNote>(con);

n.Name = one.Title;

**return** n;

}

}

}

Код JSONNote (JSONNote.cs)

**using** **System**;

**namespace** **\_4EverNote**

{

**public** **class** **JSONNote**

{

**public** **string** Content { **get**; **set**; }

**public** DateTime Created { **get**; **set**; }

**public** DateTime EventTime { **get**; **set**; }

**public** DateTime ReminderTime { **get**; **set**; }

**public** **string** Address { **get**; **set**; }

**public** **string** Reminder { **get**; **set**; }

}

}