Автоматизация приложений в среде Access

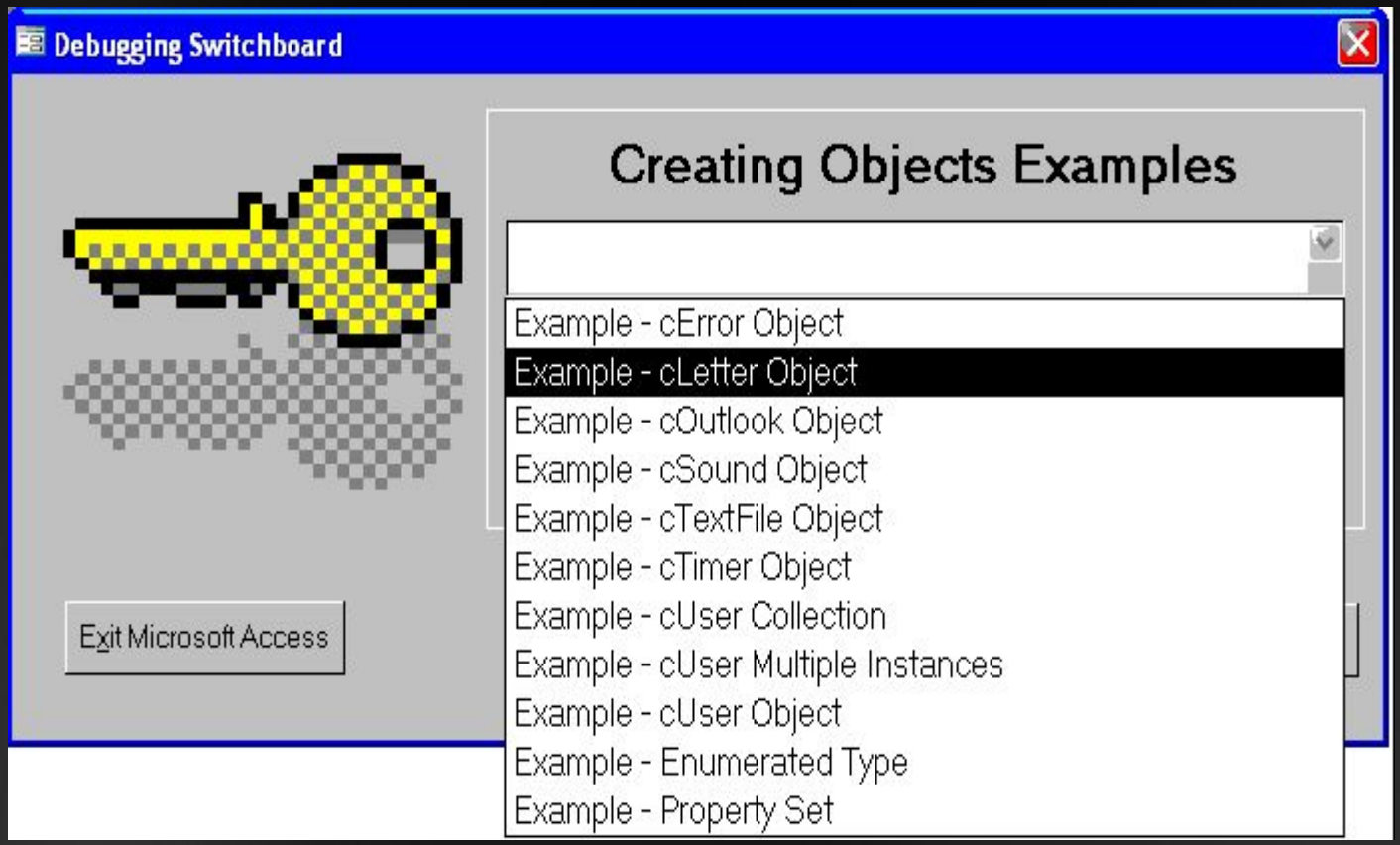
Автор : Аманжолов Н.Е.

Создание объектов с помощью модуля классов.

Создание объектов — один из самых эффективных способов программирования и управления приложениями. Перед рассмотрением преимуществ использования объектов необходимо разобраться с определениями.

Объекты представляют собой различные сущности. Люди, машины и здания — все это разные виды объектов. В программировании слово *объект* используется для описания одного специфического элемента, например, формы или элемента управления. Вне всяких сомнений, у любого разработчика имеется определенный опыт работы с этими типами встроенных объектов.

Microsoft Access приобретает все более объектно-ориентированный характер, позволяя создавать собственные объекты и добавлять к-ним свойства и методы. Примеры пользовательских объектов могут включать объект клиента, объект счета, объект пользователя, объект подключения данных и звуковой объект (см рис. 7.1).


Debugging Switchboard
Creating Objects Examples
Example - cError Object
Example - cLetter Object
Example - cOutlook Object
Example - cSound Object
Example - cTextFile Object
Example - cTimer Object
Example - cUser Collection
Example - cUser Multiple Instances
Example - cUser Object
Example - Enumerated Type
Example - Property Set
Exit Microsoft Access

Создание объектов с помощью модуля классов.

Объекты создаются с помощью модулей классов. **Модуль класса** — это мобильная, самодостаточная единица программного кода, разработанная для специальных целей. Класс указывает свойства и методы для каждого объекта в данном классе.

Разработчики часто путают термины "класс" и "объект". *Класс* представляет собой описание или шаблон свойств и методов объекта. Сам по себе он не может существовать. Если разработчику нужно использовать код в модуле класса, создается экземпляр класса. Этот экземпляр и представляет собой *объект*.

Таким образом, нельзя говорить о создании экземпляра объекта. **Объект** — это и *есть* экземпляр! Отдельный объект определяется как экземпляр класса. Из одного класса можно создать множество объектов, каждый с разными значениями свойств.

Преимущества использования объектов

Существует много преимуществ использования объектов, среди которых возможность создания сложного программного кода, использования IntelliSense, упрощенное создание и поддержка программного кода и т.д.

Сокрытие сложного программного кода

Возможность сокрытия сложного программного кода является одним из преимуществ использования объектов. Опытный разработчик может создавать сложные программы, такие как процедуры Windows API, процедуры доступа к данным, обработки строк и т.д. Менее опытные разработчики могут воспользоваться преимуществами объекта, используя вызов его свойств и методов. При этом нет необходимости разбираться в рабочем коде объекта.

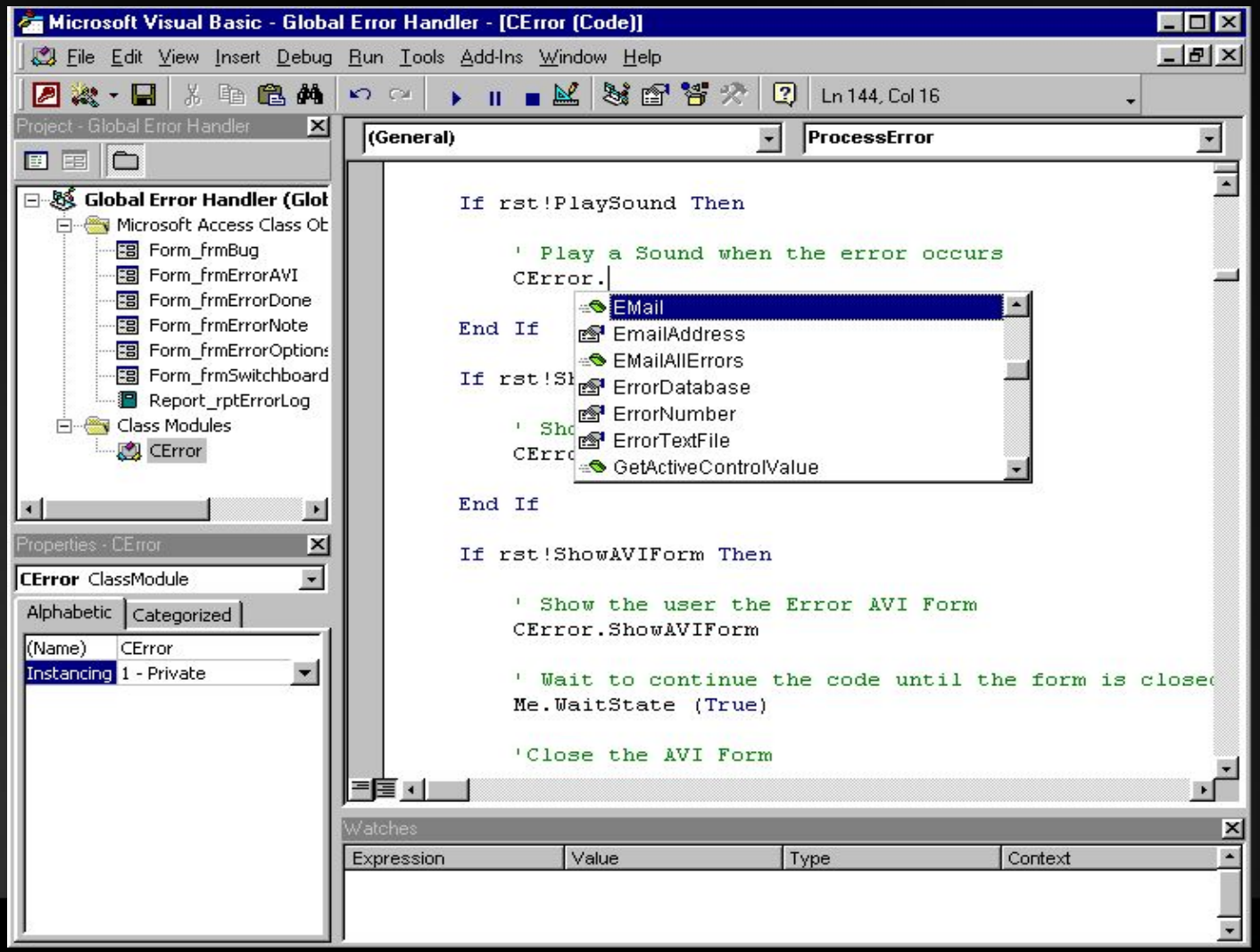
Преимущества использования объектов

Использование технологии Microsoft IntelliSense

Для использования объекта разработчику достаточно выбрать данный объект и его свойства или метод с помощью технологии IntelliSense (рис. 7.2). Например, один из объектов, рассматриваемый в данной статье, представляет собой объект обработчика ошибок, который можно использовать в приложениях Access. Поскольку весь код обработчика ошибок включен в состав объекта **cErrors,** разработчику достаточно указать объект и выбрать необходимые свойства или метод. Например, если разработчику нужно, чтобы при возникновении ошибки в приложении было отправлено почтовое сообщение, достаточно вызвать метод "email":

Преимущества использования объектов

cError.email



Организация кода

Создание классов помогает структурировать программный код, улучшая его восприятие и упрощая работу с ним. Весь программный код, выполняющий ту или иную задачу, содержится в классе. Упаковка кода в одну компактную группу со свойствами и методами, характеризующими объект, называется ***инкапсуляцией***.

Просмотр объектов в браузере объектов

Упаковка кода в классы позволяет разработчикам использовать Object Browser (Браузер объектов) для просмотра свойств, методов и другой информации. Далее в данной статье будет представлен пользовательский объект, просматриваемый с помощью браузера объектов.

Создание экземпляров объекта

Рассмотрим повторно используемый в приложении программный код. Фактически его можно использовать одновременно в нескольких экземплярах объекта. Примером может служить код подключения к данным. Каждый раз, когда пользователь осуществляет поиск покупателя, открывает форму или получает доступ к полю со списком, с сервера должны быть получены данные. Вместо того чтобы дублировать код для каждой из данных процедур, лучше создать объект **Data Connection,** позволяющий получить данные с сервера.

Упрощение кода в целях последующей поддержки и обновления

С помощью классов единственный фрагмент кода можно использовать в приложении множество раз без повтора кода. Если, например, код доступа к данным один и тот же в нескольких процедурах, необходимо найти все места, где появляется данный блок кода, и проверить, не нужно ли внести изменения. Такой подход занимает очень много времени и совершенно неэффективен. С другой стороны, если используется объект **Data Connection,** все обновления или изменения кода доступа к данным необходимо внести лишь в одном месте — модуле класса.

Ограничение доступа к коду

Классы позволяют контролировать, кто получает доступ к программному коду и при каких обстоятельствах. С помощью классов можно защитить свойства и методы, управляя их отображением вне класса.

Переносимость кода

Поскольку модуль класса является автономной единицей кода, его с легкостью можно использовать в разных приложениях Access.

Объекты, свойства и методы

Перед рассмотрением процесса создания объектов необходимо вспомнить основы. Важно понимать суть терминов *объект*, *свойства* и *методы*, иначе оставшаяся часть статьи будет весьма трудной для восприятия.

Объект — это элемент, которым можно управлять, манипулировать и который можно программировать. В Access объектами являются формы, текстовые поля, командные кнопки и многое другое. В данной статье рассматриваются примеры создания пользовательских объектов.

Объекты, свойства и методы

Свойство — это характеристика объекта. Свойство можно рассматривать и как описание чего-либо, поскольку оно описывает или характеризует объект. В Access свойства текстового поля включают **Name,** Visible, **Forecolor** и др. Большинство свойств объекта можно как считывать, так и устанавливать. В данной статье приведены примеры создания пользовательских свойств и описаны способы управления ими.

Метод представляет собой действие, которое можно выполнить над объектом. Метод можно рассматривать как действие. Например, одним из методов формы Access является метод **Close.** В данной статье приведены примеры создания пользовательских методов.

Создание классов

Для создания классов используются модули класса. Класс определяет свойства, методы и события, обеспечивая четко определенный и полностью документированный пользовательский интерфейс.

Вставка модуля класса

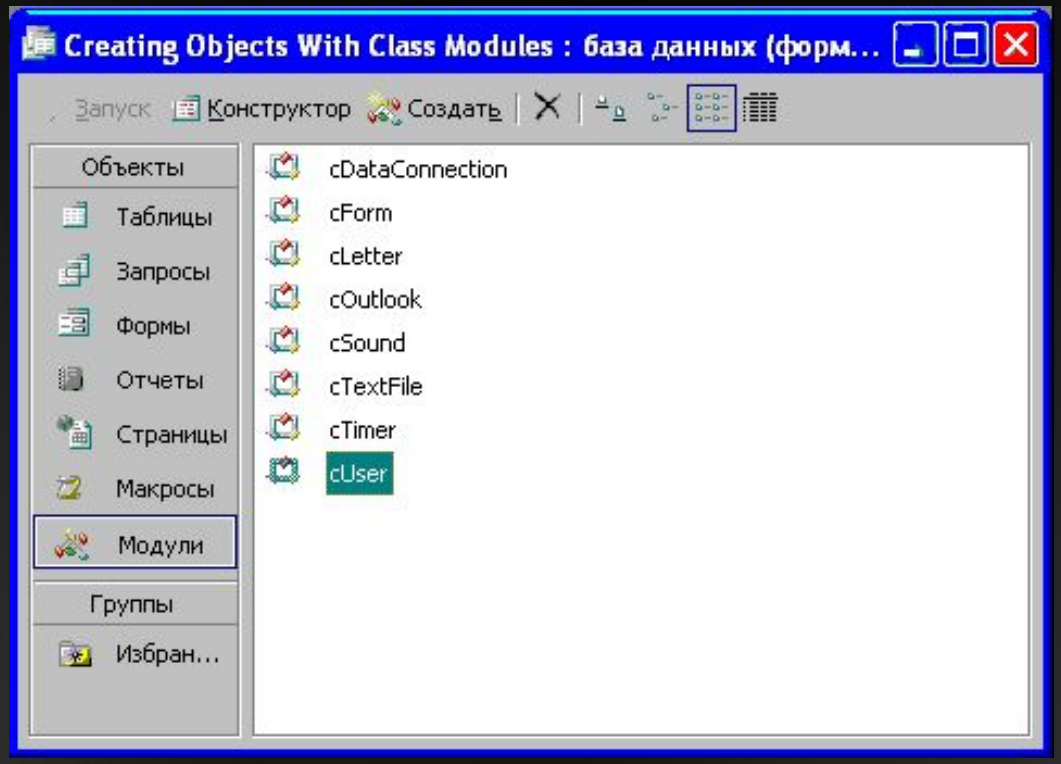
Создание класса не вызывает никаких сложностей. Добавление свойств и методов к объекту требует несколько больших усилий.

Для создания класса необходимо вставить модуль класса в приложение Access и присвоить **ему** имя. Имя модуля класса является именем объекта.

Вставка модуля класса

Чтобы вставить модуль класса, необходимо в меню выбрать пункты Insert | Class Module (Вставка | Модуль класса). После этого откроется окно редактора Visual Basic. В окне кода можно определять любые свойства, методы и события для данного объекта. Перед тем как продвигаться дальше, необходимо присвоить имя модулю класса (рис. 7.3).

Вставка модуля класса



Создание свойств

Существует два способа создания свойств класса: глобальные переменные либо процедуры свойств.

Использование общедоступных переменных

Свойство можно создать с помощью объявления общедоступной переменной в разделе **объявлений** модуля класса. В приведенном ниже примере создается свойство **UserType** объекта **cUser:**

```
Public Name as String
Public UserType as String
```

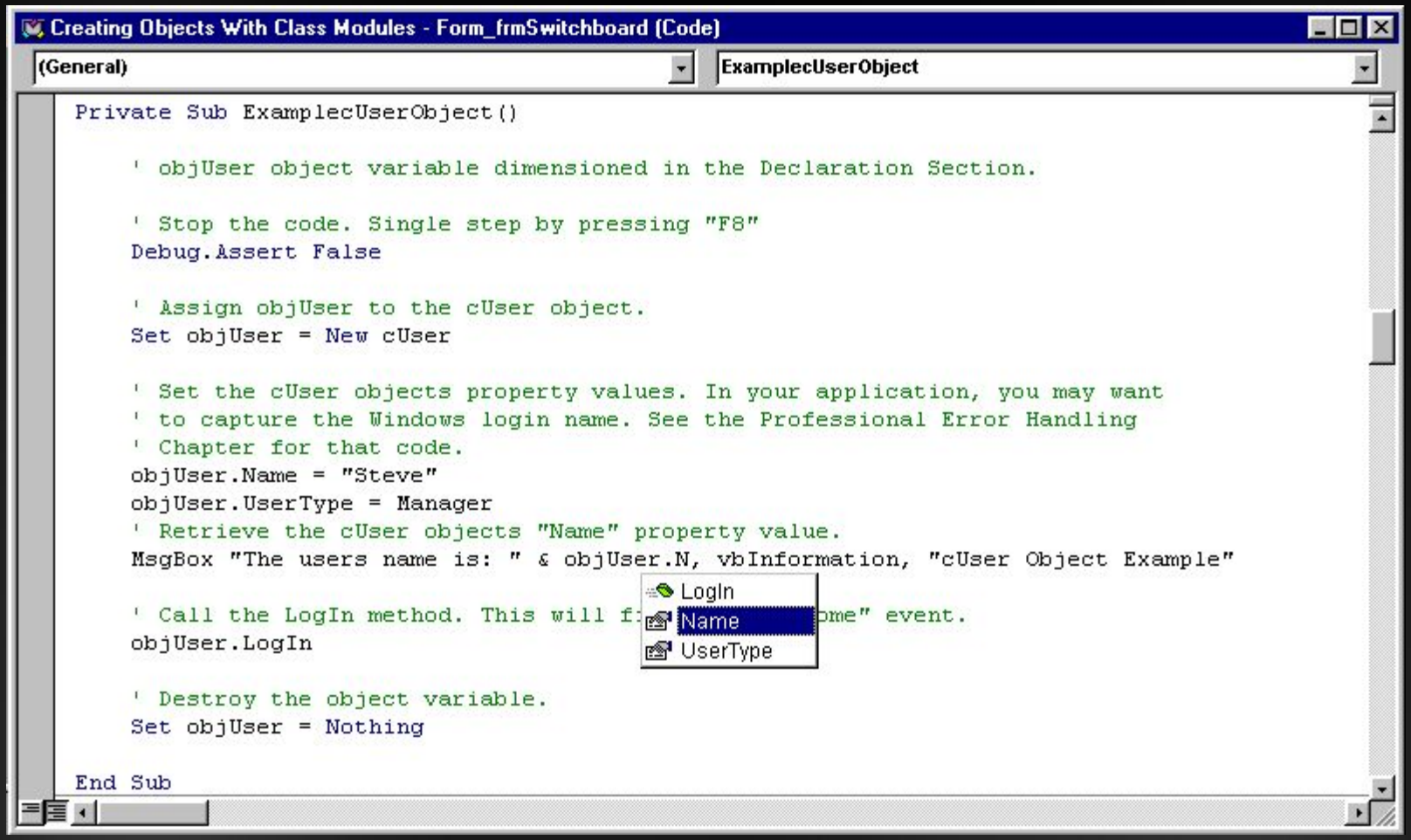
После этого несложного объявления пользователи могут устанавливать и получать значения свойства. Для установки значения **свойства применяется следующий программный код:**

```
cUser.Name = "Steve"
cUser.UserType = "Management"
```

Для выборки значения свойства используется следующий программный код (рис. 7.4);

```
MagBox cUser.Name
```

Использование общедоступных переменных



Создание приватной переменной модуля

При использовании процедур свойств значение свойства хранится в приватной переменной модуля. Создатель класса определяет, отображается ли данное свойство вне модуля.

Ниже приведен пример создания приватной переменной на уровне модуля для свойств **Name** и Type объекта cUser.

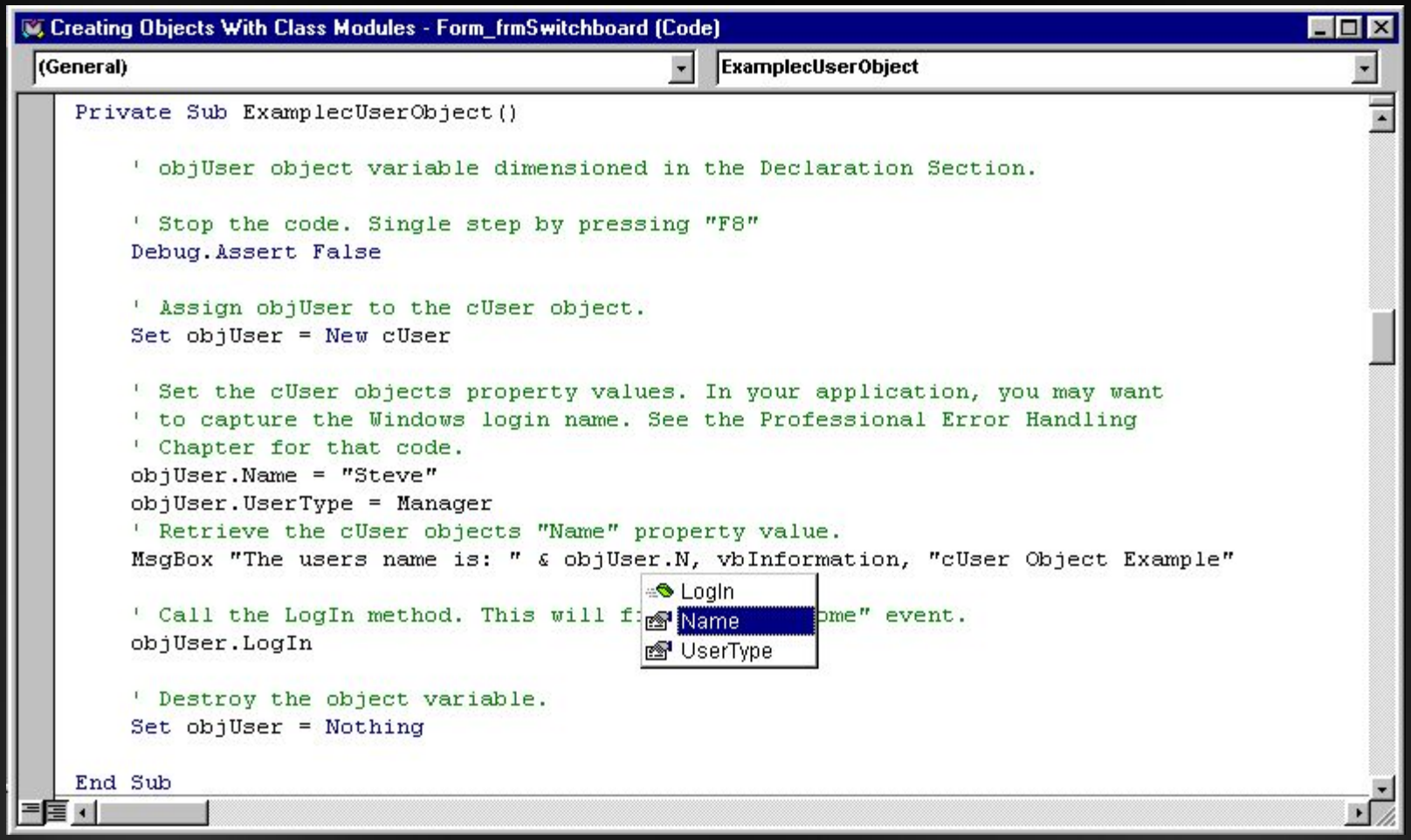
```
Option Explicit

' Создание приватной переменной в разделе объявлений.

Private mstrName as String

Private mUserType as String
```

Создание приватной переменной модуля



Процедура Property Let

Процедура **Property Let** используется для установки значения свойства. Если не нужно, чтобы другие пользователи устанавливали значение свойства, не следует включать процедуру **Property Let.** Ниже приведен пример создания процедуры **Property Let** для свойства **Name** объекта **cUser:**

```
Public Property Let Name (UserName as String)
' Принимается значение, передаваемое в UserName, и сохраняется
' в приватной переменной (mstxName).
mstrName = UserName
End Property
```

Рассмотрим данную процедуру подробнее. Во-первых, поскольку существует процедура Property **Let,** можно устанавливать свойство Name, так как данное свойство отображается вне модуля. Разработчик мог бы присвоить этому свойству следующее значение:

```
cUser.Name = "James"
```

Процедура Property Let

Значение **James,** передаваемое в процедуру свойства, сохраняется в переменной **UserName.** Процедура **Property Let** принимает значение переменной **UserName** (James) и записывает его в приватной переменной модуля **mstrName.** Рассматриваемая процедура передает один параметр, хотя на самом деле процедуры свойств могут передавать много параметров. Значение свойства может быть получено только в том случае, когда существует процедура **Property Get.**

Процедура Property Get

Процедура **Property Get** позволяет получить значение свойства. Если не нужно, чтобы другие пользователи могли получить значение свойства, не следует включать оператор **Property Get.** Оператор **Property Get** получает значение, хранящееся в приватной переменной, и возвращает его в качестве значения свойства. Приведенный ниже пример представляет собой оператор **Property Get** для свойства **Name** объекта **cUser:**

```
Public Property Get Name () as String
' Получение значения, записанного в приватной переменной (mstrName),
' и запись его в значение свойства.
Name = mstrName
End Property
```

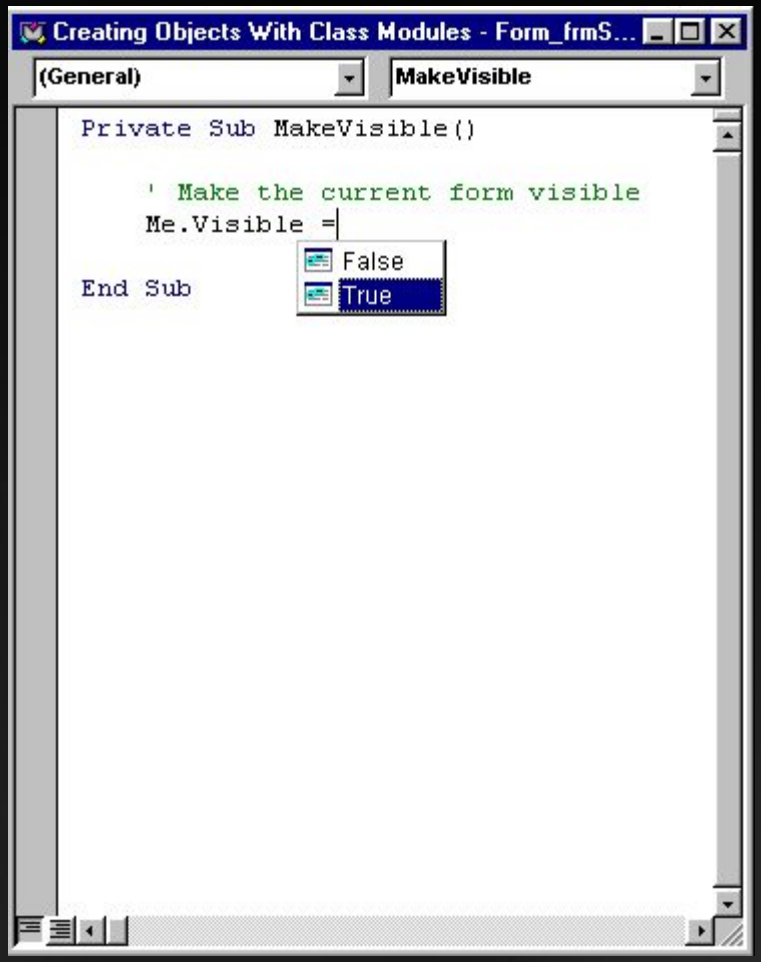
Пользователь легко может получить значение свойства (если существует выражение **Property Get),** воспользовавшись следующим кодом:

```
MsgBox cUser.Name
```

Общедоступная переменная и процедуры свойств

Простейший способ создания свойств класса заключается в использовании общедоступных переменных. Однако данный подход имеет некоторые недостатки. Общедоступные переменные всегда отображаются вне модуля. Таким образом, невозможно контролировать доступ к значениям свойств. Если кто-либо меняет значения свойств без ведома других пользователей, это может вызвать нежелательные проблемы.

Общедоступная переменная и процедуры свойств



Общедоступная переменная и процедуры свойств

Еще одно преимущество процедур свойств заключается в том, что можно создать свойства "только для чтения" или "только для записи". Например, если нужно создать свойство **Password,** можно разрешить пользователям устанавливать пароль, но не получать его (только для записи). Для создания свойства "только для записи" необходимо включить выражение **Property Let,** но не включать выражение **Property Get.**

Кроме того, с помощью процедур свойств можно выполнять действия в коде в зависимости от того, установлено свойство или получено.

Перечисляемые типы данных

Перечисляемый тип — это значение свойства, которое может быть передано разработчику при использовании объекта. Например, при установке свойства **Visible** для формы после ввода знака равенства можно заметить, что возможен выбор значений **True** и **False** в раскрывающемся списке

Создание методов

Метод — это действие, которое может быть выполнено над объектом. Чтобы создать метод для класса, достаточно создать общедоступную подпрограмму или функцию. Предположим, что нужно отслеживать дату и время каждой регистрации пользователя. Приведенный ниже код создает метод **Login**, который вводит в таблицу дату и время регистрации пользователя.

Создание методов

```
Public Sub Login()

' Создание объектной переменной для набора записей.
Dim rst As ADODB.Recordset

' Создание строковой переменной.
Dim strSQL As String

' SQL-оператор для tbIUsers.
StrSQL = "SELECT * FROM tbIUsers“

' Создание набора записей ADO.
Set rst = New ADODB.Recordset

' Открытие набора записей.
rst.Open strSQL, CurrentProject.Connection, adOpenKeyset, adLockOptimistic
```

Создание методов

```
' Добавление новой записи.
rstAddNew

' Запись даты и времени регистрации пользователя.
With rst
!Name = Me.Name
!Date = Date
!Time = Time End With

' Сохранение новой записи.
rst.Update

' Закрытие набора записей.
rst.Close

' Уничтожение объектной переменной.
Set rst = Nothing
End Sub
```

Использование методов

Для использования данного метода при запуске приложения необходимо ввести код:

cUser.login

Обратите внимание, что пользователям, которым необходимо обновить значение в базе данных, не обязательно разбираться в коде ADO. Они просто используют доступные свойства и методы объекта, которые отображаются с помощью технологии IntelliSense.

Создание событий

Объекты Access включают события. Например, объект формы содержит событие **Load,** а командная кнопка — событие **Click.**

Можно создавать события для пользовательских объектов. Для этого в разделе объявлений необходимо воспользоваться ключевым словом Event и указать имя события. Например, можно добавить событие Welcome, которое выполняется при запуске приложения пользователем. Для создания данного события в разделе объявлений модуля класса необходимо ввести следующий код:

```
Event Welcome()
```

Чтобы воспользоваться событием, его необходимо сформировать с помощью инструкции **Raise в** методе объекта. Событие **Welcome** можно сформировать в методе **Login** объекта **cUser.**

Создание событий

Когда пользователь запускает приложение, вызывается метод **Login.** Благодаря этому вызывается событие **Welcome,** которое отображает всплывающий экран с персональным приветствием данному пользователю. Используется следующий программный код:

```
Public Sub Login()

RaiseEvent Welcome

End Sub
```

Это все, что нужно предпринять для создания события и формирования его в модуле класса. В следующем разделе описано использование события в форме.

Создание событий

Использование событий

Чтобы воспользоваться событием **Welcome в** модуле формы, необходимо объявить объектную переменную для **cUser** в разделе объявлений модуля и использовать ключевое слово **WithEvents.** Зарезервированное слово **WithEvents** применяется для отображения и использования событий объекта **cUser**. Соответствующий код приведен ниже:

```
' Данное выражение следует поместить в раздел объявлений модуля.
Private WithEvents objUser As cUser
```

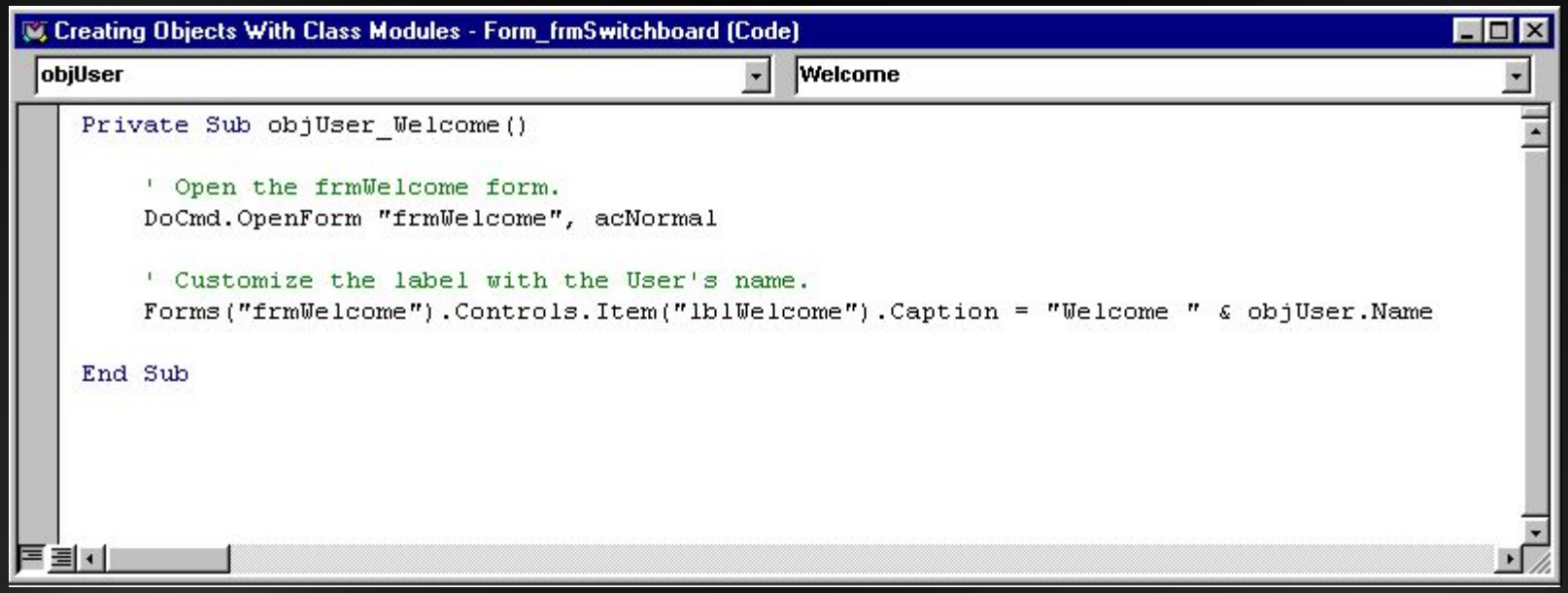
Создание событий

После объявления переменной на уровне модуля с помощью ключевого слова **WithEvents** необходимо в левом поле со списком в верхней части окна модуля выбрать объект **cUser.** В правом поле со списком будут отображены события объекта **cUser.** Выберите событие **Welcome,** и в окне кода появится процедура, в которой можно записать соответствующий код для ответа на событие.

На рис. 7.7 приведен код, отвечающий на событие **Welcome.**

Создание событий

Рис. 7.7



Вызов событий Initialize и Terminate

Модули класса автоматически включают события инициализации и завершения. Для использования данных событий необходимо выбрать событие в поле со списком в верхней части окна кода.

Событие инициализации вызывается при создании объекта. Например, если нужно, чтобы выполнялся определенный код при создании объекта **cUser,** данный код можно поместить в событие **Initialize.**

Событие завершения происходит при разрушении объекта. Здесь удобно записывать код очистки, выполняющийся при закрытии подключений к базам данных, освобождении объектных переменных и т.д.

Использование объектов

На данном этапе существует объект **cUser** со своими свойствами и методами. Чтобы использовать его в модуле класса, достаточно ввести имя объекта и добавить в конце точку. IntelliSense отобразит список свойств и методов объекта.

Для использования объекта в классе, который создает объект, употребляется ключевое слово **Me.** Например, чтобы присвоить имя пользователя, необходимо ввести:

Me.User = "Steve“

Использование объекта вне модуля класса **cUser** (в формах либо стандартных модулях) разбивается на два этапа. Во-первых, необходимо определить объектную переменную для использования в качестве ссылки на объект. Во-вторых, используется ключевое слово **Set** для создания ссылки из объектной переменной на объект.

Создание объектной переменной

Переменная — это участок памяти, отведенный для хранения или использующийся при считывании информации. Вне всяких сомнений, у разработчиков имеется громадный опыт работы с простыми переменными, такими как строковые и целочисленные переменные. Ниже приведены примеры объявления и использования двух простых переменных:

```
Dim sfcrNaor aa String
Dim I as integer
strNama = "James"
I = 10
```

В этих примерах переменные включают специфический тип данных, и информация может храниться и считываться по необходимости.

Переменная **Object** объявляется с помощью выражения **Dim**, как и простые переменные:

```
Dim objUser as cUser
```

Присваивание объектной переменной объекту

Для установки ссылки объектной переменной на объект используется ключевое слово **Set**. Например:

```
Set objUser = New cUser
```

Использование объекта

После создания и установки объектной переменной доступ к свойствам и методам объекта можно получить с помощью технологии IntclliSense. Используя точечную нотацию, можно установить и считать значение свойства, а также выполнить метод.

Установка значения свойства: **Object.Property = Value**

Получение значения свойства: **MsgBox Object.Property**

Освобождение объекта

Закончив работу с объектной переменной, ее необходимо освободить, установив значение объекта равным Nothing. Таким образом можно высвободить ценные ресурсы. Соответствующий синтаксис приведен ниже:

```
' Освобождение объектной переменной.
Set objUser = Nothing
```

Создание нескольких экземпляров объекта

Класс содержит определенный набор свойств и методов. Этим он напоминает шаблон или каркас. Для иллюстрации рассмотрим открытие нового документа в Microsoft Word.

При создании нового документа Word можно воспользоваться шаблоном, например. Contemporary Letter (Современное письмо). После выбора шаблона в документ можно внести изменения. Данные изменения вносятся в новый документ, а *не* в шаблон. Тот же самый принцип действует в случае с классами.

Создание нескольких экземпляров объекта

При инициализации (создании) класса экземпляр объекта содержит базовый набор свойств и методов (шаблон). С этого момента объект становится уникальным, обладая собственным набором значений свойств и методов.

Одно из преимуществ модулей класса заключается в том, что можно создать много экземпляров класса. Каждый экземпляр создастся с базовым набором свойств и методов, но после создания каждый из них может быть изменен различным образом. Например, предположим, создано пять объектов **cUser.** Вес пять объектов могут иметь разные значения свойств и разные методы. Например, значение свойства **Name в** одном объекте может быть **James,** а **в** другом — **Steve.**

Создание нескольких экземпляров объекта

Для создания нескольких экземпляров объекта достаточно создать дополнительные объектные переменные и присвоить каждую объектную переменную новому объекту. Соответствующий код приведен ниже.

Создание нескольких экземпляров объекта

```
Public Sub ManyUsersO
Dim objUser1 as cUser
Dim objUser2 as cUser
Set objUser1 = New cUser
Set objUser2 = New cUser
ObjUser1.Name = "James"
ObjUser2.Name = "Steve"
MsgBox "Current users are: " & objUser1.Name & " and " objUser2.Name
Set objUser1 = Nothing
Set objUser2 = Nothing
End Sub
```