



Наследование

Последнее обновление: 30.10.2015



Одним из ключевых аспектов объектно-ориентированного программирования является **наследование (inheritance)**. Сущность наследования заключается в том, что мы можем расширить функционал уже имеющихся классов с помощью создания наследников этого класса. Допустим, у нас определен следующий класс Person, описывающий отдельного человека:

```
1 Public Class Person
2     Dim _firstName As String
3     Dim _lastName As String
4
5     Public Property FirstName() As String
6         Get
7             Return _firstName
8         End Get
9         Set(value As String)
10            _firstName = value
11        End Set
12    End Property
13
14    Public Property LastName() As String
15        Get
16            Return _lastName
17        End Get
18        Set(value As String)
19            _lastName = value
20        End Set
21    End Property
22
23    Public Sub Display()
24        Console.WriteLine(FirstName & " " & LastName)
25    End Sub
26 End Class
```

Теперь создадим класс, описывающий сотрудника предприятия - класс **Employee**.

Поскольку этот класс будет реализовывать тот же функционал, что и класс Person, так

как сотрудник – это также и человек, то было бы рационально сделать класс **Employee** производным (или наследником) от класса **Person**, который, в свою очередь, называется базовым классом или родителем. Чтобы наследовать один класс от другого, нужно использовать ключевое слово **Inherits**:

```
1 Public Class Employee
2     Inherits Person
3
4 End Class
```

По умолчанию все классы могут наследоваться. Однако здесь есть ряд ограничений:

- Во-первых, не поддерживается двойное наследование, класс может наследоваться только от одного класса. Хотя проблема множественного наследования реализуется с помощью концепции интерфейсов, о которых мы поговорим позже.
- Во-вторых, при создании производного класса мы должны учитывать тип доступа к базовому классу – тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть если базовый класс у нас имеет тип доступа **Friend**, то производный класс может иметь тип доступа **Friend** или **Private**, но не **Public**.
- В-третьих, если класс объявлен с модификатором **NotInheritable**, то от этого класса нельзя наследовать и создавать производные классы. Например, следующий класс не допускает создание наследников:

```
1 Public NotInheritable Class Person
2
3 End Class
```

И в этом случае создать от этого класса класс **Employee** как и любой другой мы не сможем.

Но вернемся к нашему классу **Employee**. Поскольку он образован от класса **Person**, то он обладает той же функциональностью, что определена в классе **Person**. Однако посмотрим, что будет, если мы захотим создать метод для вывода на экран имени сотрудника, которое храниться в переменной `_firstName`:

```
1 Public Class Employee
2     Inherits Person
3     Public Sub DisplayFirstName()
4         Console.WriteLine(_firstName)
5     End Sub
6 End Class
```

Этот код у нас работать не будет, так как переменная `_firstName` определена как **Private** (как вы помните, модификатор `Dim` аналогичен модификатору `Private`), поэтому

к ней может иметь доступ только класс `Person`. Но зато в классе `Person` определено общедоступное свойство `FirstName`, которое мы можем использовать, поэтому следующий код у нас будет работать нормально:

```
1 Public Class Employee
2     Inherits Person
3     Public Sub DisplayFirstName()
4         Console.WriteLine(FirstName)
5     End Sub
6 End Class
```

Таким образом, производный класс может иметь доступ только к тем членам базового класса, которые определены с модификаторами **Public**, **Friend**, **Protected** и **Protected Friend**.

Переопределение методов в производных классах

Итак, мы уже можем использовать наши классы в программе:

```
1 Sub Main()
2
3     Dim emp As New Employee()
4     emp.FirstName = "Bill"
5     emp.LastName = "Gates"
6     emp.Display()
7
8     Console.ReadLine()
9 End Sub
```

Но что если мы захотим добавить в класс `Employee` свойство, представляющее компанию, в которой сотрудник работает, и выводить в методе `Display` сведения о компании. То есть мы хотим переопределить функционал метода `Display` в производном классе. Для этого нам надо в классе `Person` пометить метод `Display` модификатором **Overridable**. А уже в производном классе этот же метод использовать с модификатором **Overrides**:

```
1 'Базовый класс Person
2 Public Class Person
3     'Здесь остальной код
4     .....
5     Public Overridable Sub Display()
6         Console.WriteLine(FirstName & " " & LastName)
7     End Sub
8 End Class
9
10 'Производный класс Employee
11 Public Class Employee
12     Inherits Person
```

```
13
14     Public Property Company As String
15
16     Public Overrides Sub Display()
17         Console.WriteLine(FirstName & " " & LastName & " works in " & Company)
18     End Sub
19 End Class
```

Чтобы запретить переопределение, вместо **Overridable** надо использовать ключевое слово **NotOverridable** (оно используется по умолчанию, если не указано **Overridable**).

Следует сказать, что мы могли обойтись в нашем примере и без переопределения. Мы могли просто скрыть метод базового класса и объявить в производном классе новый метод с таким же именем. Для этого в объявлении метода в производном классе надо использовать ключевое слово **Shadows**:

```
1  'Базовый класс Person
2  Public Class Person
3      'Здесь остальной код
4      .....
5      Public Sub Display()
6          Console.WriteLine(FirstName & " " & LastName)
7      End Sub
8  End Class
9
10 'Производный класс Employee
11 Public Class Employee
12     Inherits Person
13
14     Public Property Company As String
15
16     Public Shadows Sub Display()
17         Console.WriteLine(FirstName & " " & LastName & " works in " & Company)
18     End Sub
19 End Class
```

И если мы вызовем метод `Display` у объекта класса `Employee`, то будет использоваться именно метод с модификатором **Shadows**:

```
1  Sub Main()
2
3      Dim emp As New Employee()
4      emp.FirstName = "Bill"
5      emp.LastName = "Gates"
6      emp.Company = "Microsoft"
7      emp.Display() 'Здесь будет выведено 'Bill Gates works in Microsoft'
8
9      Console.ReadLine()
10 End Sub
```

При этом важно помнить, что **Shadows** не переопределяет метод базового класса, а лишь скрывает его.

Ключевое слово MyBase.

Теперь добавим в наши классы конструкторы: один конструктор будет без параметров и один с параметрами:

```
1  'Базовый класс Person
2  Public Class Person
3      'Здесь остальной код
4      .....
5      'Конструкторы класса
6      Public Sub New()
7
8      End Sub
9
10     Public Sub New(fName As String, lName As String)
11         FirstName = fName
12         LastName = lName
13     End Sub
14 End Class
15
16 'Производный класс Employee
17 Public Class Employee
18     Inherits Person
19
20     'Здесь остальной код
21     .....
22     'Конструкторы класса
23     Public Sub New(fName As String, lName As String, comp As String)
24         MyBase.New(fName, lName)
25         Company = comp
26     End Sub
27
28     Public Sub New()
29
30     End Sub
31 End Class
```

В конструктор с параметрами мы передаем значения для свойств класса. В конструкторе класса `Person` мы устанавливаем имя и фамилию объекта, а в конструкторе класса `Employee` - имя, фамилию и компанию. Поскольку при создании объекта класса вызываются все конструкторы его родительских классов, то есть при создании объекта класса `Employee`:

```
1  Dim emp As New Employee()
```

Сначала вызывается конструктор без параметров класса `Person`, а затем уже конструктор без параметров класса `Employee`. Поэтому нам нет смысла устанавливать все свойства в конструкторе `Employee`. И мы передаем часть значений в конструктор родительского класса с помощью ключевого слова **MyBase**. Это слово используется для получения членов родительского класса. В данном случае мы получили доступ к конструктору.

[Назад](#) [Содержание](#) [Вперед](#)



ТАКЖЕ НА METANIT.COM

Взаимодействие с кодом Python

5 месяцев назад · 4 коммен...

Взаимодействие с кодом Python в программе на языке Си, установка ...

Встроенные компоненты ввода

5 месяцев назад · 1 коммен...

Встроенные компоненты ввода Blazor из пространства имен ...

Ассемблер MASM. Установка и ...

3 месяца назад · 4 коммент...

Ассемблер MASM. Установка и начало работы, Visual Studio, ...

ListVi

2 меся

ListVi
созда
получ

2 Комментариев

1 Войти ▾

Присоединиться к обсуждению...

ВОЙТИ С ПОМОЩЬЮ

ИЛИ ЧЕРЕЗ DISQUS 

Имя



Поделиться

Лучшие

Новые

Старые

S

Sergey Gorikov

год назад

Salute, спс за материал:)

При нажатии "Вперед" со странички
"visualbasic/tutorial/3.6.php" –Наследование
осуществляется переход на страничку
"visualbasic/tutorial/3.8.php" –Интерфейсы

*Похоже что пропущена глава
"visualbasic/tutorial/3.7.php" –Абстрактные классы

1 0 Ответить • Поделиться ›



Metanit Модератор

➔ Sergey Gorikov

год назад

Помощь сайту

YooMoney:

41001174743222

Qiwi:

qiwi.com/n/METANIT

Перевод на карту

Номер карты:

4048415020898850

[Вконтакте](#) | [Телеграм](#) | [Twitter](#) | [Помощь сайту](#)

Контакты для связи: metanit22@mail.ru

Copyright © metanit.com, 2023. Все права защищены.

