



## Преобразование типов

Последнее обновление: 30.10.2015



Ранее мы уже говорили о преобразовании простейших типов. Теперь затронем тему преобразования объектов различных классов. Итак, у нас есть следующая система классов:

```
1  'Классы
2  Public MustInherit Class Person
3
4      Public Property FirstName() As String
5      Public Property LastName() As String
6      'Абстрактный метод
7      Public MustOverride Sub Display()
8
9      Public Sub New(fName As String, lName As String)
10         FirstName = fName
11         LastName = lName
12     End Sub
13
14 End Class
15
16 Public Class Employee
17     Inherits Person
18
19     Public Property Bank As String
20
21     Public Overrides Sub Display()
22         Console.WriteLine(FirstName & " " & LastName & " works in " & Bank)
23     End Sub
24
25     Public Sub New(fName As String, lName As String, _bank As String)
26         MyBase.New(fName, lName)
27         Bank = _bank
28     End Sub
29
30 End Class
31
32 Public Class Client
```

```
33 Inherits Person
34 Implements IAccount
35
36 'Переменная для хранения суммы
37 Dim _sum As Integer
38 'Переменная для хранения процента
39 Dim _procentage As Integer
40
41 Public Property Bank As String
42
43 'Текущая сумма на счете
44 ReadOnly Property CurentSum() As Integer Implements IAccount.CurentSum
45     Get
46         Return _sum
47     End Get
48 End Property
49 'Метод для добавления денег на счет
50 Sub Put(sum As Integer) Implements IAccount.Put
51     _sum += sum
52 End Sub
53 'Метод для снятия денег со счета
54 Sub Withdraw(sum As Integer) Implements IAccount.Withdraw
55     If sum <= CurentSum Then
56         _sum -= sum
57     End If
58 End Sub
59 'Процент начислений
60 ReadOnly Property Procentage() As Integer Implements IAccount.Procentage
61     Get
62         Return _procentage
63     End Get
64 End Property
65
66 Public Overrides Sub Display()
67     Console.WriteLine(FirstName & " " & LastName & " has an account in b
68 End Sub
69
70 Public Sub New(fName As String, lName As String, _bank As String, _sum As Integer)
71     MyBase.New(fName, lName)
72     Bank = _bank
73     Me._sum = _sum
74 End Sub
75
76 End Class
77
78 'Интерфейсы
79 Public Interface IAccount
80     'Текущая сумма на счете
81     ReadOnly Property CurentSum() As Integer
```

```

82     'Метод для добавления денег на счет
83     Sub Put(sum As Integer)
84     'Метод для снятия денег со счета
85     Sub Withdraw(sum As Integer)
86     'Процент начислений
87     ReadOnly Property Procentage() As Integer
88 End Interface
89
90 Public Interface IDepositAccount
91     Inherits IAccount
92
93     'Начисление процентов
94     Sub GetIncome()
95
96 End Interface

```

В иерархии классов мы можем проследить следующую цепь наследования: Object (все классы неявно наследуются от типа Object) -> Person -> Employee|Client.

Из этой цепи видно, что объект класса Client одновременно является и представителем классов Person и Object. Поэтому мы можем создать объекты обоих классов следующим образом:

```

1 Dim emp As Object = New Employee("John", "Thompson", "City Bank")
2 Dim cl As Person = New Client("Tom", "Johnson", "City Bank", 100)
3
4 emp.Display()
5 cl.Display()

```

В данном случае работает неявное преобразование, так как наши переменные представляют классы Object и Person, поэтому допустимо неявное восходящее преобразование - преобразование к типам, которые находятся вверху иерархии классов:

```

Object
|
Person
|
Employee|Client

```

Таким образом, классы Employee и Client находятся в низу иерархии классов. Добавим еще класс **Manager**, который будет производным от Employee и поэтому будет находиться в самом низу иерархии классов.

```

1 Public Class Manager
2     Inherits Employee
3
4     Public Overrides Sub Display()
5         Console.WriteLine(FirstName & " " & LastName & " is a manager of " &

```

```
6 End Sub
7
8 Public Sub New(fName As String, lName As String, _bank As String)
9     MyBase.New(fName, lName, _bank)
10 End Sub
11
12 End Class
```

Значит, мы можем написать следующее, так как объект класса Manager в то же время является и сотрудником банка:

```
1 Dim man As Employee = New Manager("John", "Thompson", "City Bank")
2 man.Display()
```

В то же время, если мы применим нисходящее преобразование, как в следующем коде, то мы получим ошибку:

```
1 Dim emp As Manager = New Employee("John", "Thompson", "City Bank")
2 emp.Display()
```

Здесь мы преобразуем объект Employee к типу Manager, однако если Manager является объектом типа Employee, то объект Employee не является объектом типа Manager. Есть несколько способов избежать ошибки при выполнении программы. Во-первых, можно использовать оператор **TryCast** - он пытается преобразовать выражение к определенному типу, при этом не выбрасывает исключение. В случае неудачного преобразования выражение будет содержать значение **Nothing**:

```
1 Dim emp As New Employee("John", "Thompson", "City Bank")
2 Dim man As Manager = TryCast(emp, Manager)
3 If man Is Nothing Then
4     Console.WriteLine("Преобразование прошло неудачно")
5 Else
6     man.Display()
7 End If
```

Второй способ заключается в отлавливании исключения **InvalidCastException**, которое возникнет в результате преобразования:

```
1 Try
2     Dim emp As New Employee("John", "Thompson", "City Bank")
3     Dim man As Manager = DirectCast(emp, Manager)
4 Catch ex As InvalidCastException
5     Console.WriteLine("Преобразование прошло неудачно")
6 End Try
```

Третий способ заключается в проверке допустимости преобразования с помощью ключевого слова **Is**:

```
1 Dim emp As New Employee("John", "Thompson", "City Bank")
```

```
2 If TypeOf emp Is Manager Then
3     Dim man As Manager = DirectCast(emp, Manager)
4 Else
5     Console.WriteLine("Преобразование недопустимо")
6 End If
```

Выражение `TypeOf emp Is Manager` указывает, имеет ли переменная `emp` тип `Manager`. А поскольку такая проверка вернет `False`, то преобразование не сработает. Для явного преобразования типов используется оператор `DirectCast`, который пытается преобразовать переменную `emp` к типу `Manager`.

Все сказанное выше в отношении преобразования типов характерно и для интерфейсов. Поскольку класс `Client` реализует интерфейс `IAccount`, то переменная типа `IAccount` может хранить ссылку на объект типа `Client`:

```
1 Dim cl As IAccount = New Client("John", "Thompson", "City Bank", 200)
2 cl.Put(200)
3 Console.WriteLine("Остаток на счете : {0}", cl.CurentSum)
```

[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 Qt, ...

### Параметры строки запроса

5 месяцев назад · 1 коммен...

Параметры строки запроса query string в приложении Blazor на ...

### ListView

2 месяца назад · 1 коммент...

ListView в JavaFX, создание списков, получение выбранных в ...

### Клиент Form

10 дне

Клиент на Xamarin Signal

0 Комментариев

1

 Войти ▼

G

Начать обсуждение...

войти с помощью

или через DISQUS 

?

Имя



Поделиться

Лучшие Новые Старые

Прокомментируйте первым.

Подписаться

О защите персональных данных

Помощь сайту

**YooMoney:**  
410011174743222

**Qiwi:**  
[qiwi.com/n/METANIT](https://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.