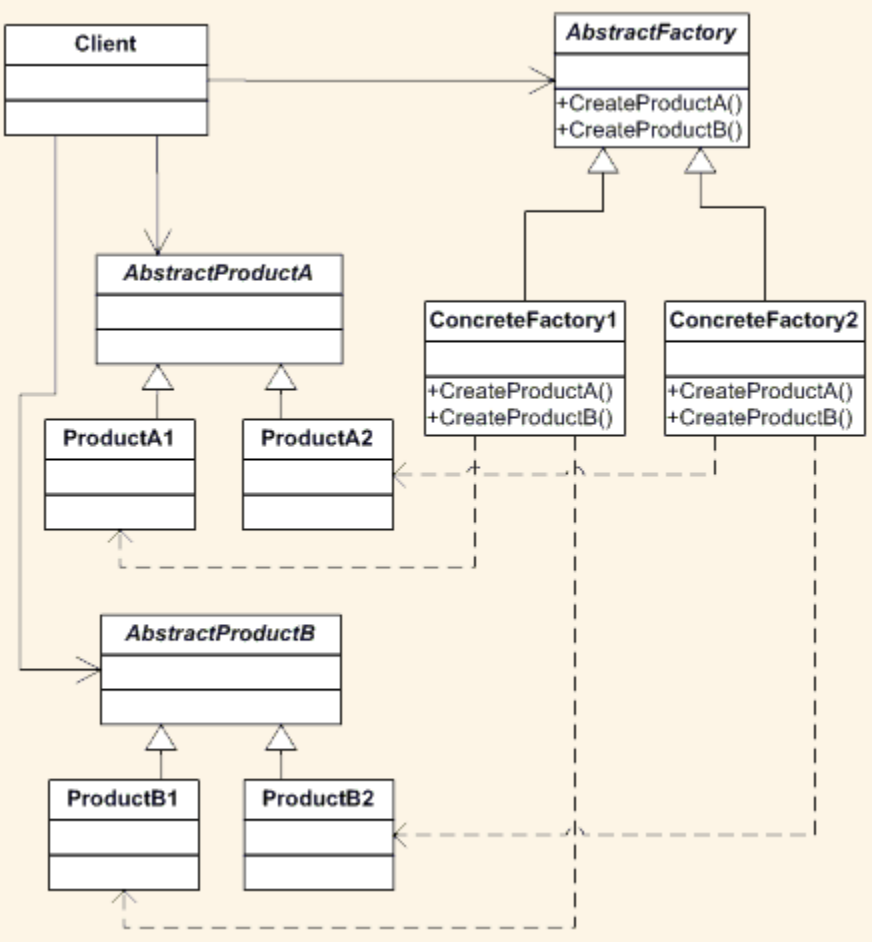
(NET) NET (2016)

Abstract Factory in VB.NET

Creates an instance of several families of classes

назад

Provide an interface for creating families of related or dependent objects without specifying their concrete classes. Object creation has been abstracted and there is no need for hard-coded class names in the client code.



3.3.1 Абстрактная фабрика (Abstract Factory, Factory), др. название Инструментарий (Kit) - GoF

Проблема	Создать семейство взаимосвязанных или взаимозависимых обьектов (не специфицируя их конкретных классов).
Решение	Создать абстрактный класс, в котором объявлен интерфейс для создания конкретных классов.
Пример	Какой класс должен отвечать за создание обьектов - адаптеров при использовании паттерна "Адаптер", см. 3.1.1. Если подобные объекты создаются неким объектом уровня предметной области, то будет нарушен принцип разделения обязанностей.
Преимущества	Изолирует конкретные классы. Поскольку "Абстрактная фабрика" инкапсулирует ответственность за создание классов и сам процесс их создания, то она изолирует клиента от деталей реализации классов. Упрощена замена "Абстрактной фабрики", поскольку она используется в приложении только один раз при инстанцировании.
Недостатки	Интерфейс "Абстрактной фабрики" фиксирует набор обьектов, которые можно создать. Расширение "Абстрактной фабрики" для изготовления новых обьектов часто затруднительно.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
' Abstract Factory Design Pattern.
' See description in http://www.vb-net.ru/ProgramTheory/AbstractFactory.htm
Class MainApp
    ' Entry point into console application.
    Public Shared Sub Main()
        ' Abstract factory #1
        Dim factory1 As AbstractFactory = New ConcreteFactory1()
        Dim client1 As New Client(factory1)
        client1.Run()
        ' Abstract factory #2
        Dim factory2 As AbstractFactory = New ConcreteFactory2()
        Dim client2 As New Client(factory2)
        client2.Run()
        ' Wait for user input
        Console.ReadKey()
    End Sub
End Class

' The 'AbstractFactory' abstract class
MustInherit Class AbstractFactory
    Public MustOverride Function CreateProductA() As AbstractProductA
    Public MustOverride Function CreateProductB() As AbstractProductB
End Class

' The 'ConcreteFactory1' class
Class ConcreteFactory1
    Inherits AbstractFactory
    Public Overrides Function CreateProductA() As AbstractProductA
        Return New ProductA1()
    End Function
    Public Overrides Function CreateProductB() As AbstractProductB
        Return New ProductB1()
    End Function
End Class

' The 'ConcreteFactory2' class
Class ConcreteFactory2
    Inherits AbstractFactory

    Public Overrides Function CreateProductA() As AbstractProductA
        Return New ProductA2()
    End Function

    Public Overrides Function CreateProductB() As AbstractProductB
        Return New ProductB2()
```

```
46:      End Function
47:  End Class
48:
49:  ' The 'AbstractProductA' abstract class
50:  MustInherit Class AbstractProductA
51:  End Class
52:
53:  ' The 'AbstractProductB' abstract class
54:  MustInherit Class AbstractProductB
55:      Public MustOverride Sub Interact(a As AbstractProductA)
56:  End Class
57:
58:  ' The 'ProductA1' class
59:  Class ProductA1
60:      Inherits AbstractProductA
61:  End Class
62:
63:  ' The 'ProductB1' class
64:  Class ProductB1
65:      Inherits AbstractProductB
66:      Public Overrides Sub Interact(a As AbstractProductA)
67:          Console.WriteLine(Me.[GetType]().Name + " interacts with " + a.[GetType]().Name)
68:      End Sub
69:  End Class
70:
71:  ' The 'ProductA2' class
72:  Class ProductA2
73:      Inherits AbstractProductA
74:  End Class
75:
76:  ' The 'ProductB2' class
77:  Class ProductB2
78:      Inherits AbstractProductB
79:      Public Overrides Sub Interact(a As AbstractProductA)
80:          Console.WriteLine(Me.[GetType]().Name + " interacts with " + a.[GetType]().Name)
81:      End Sub
82:  End Class
83:
84:  ' The 'Client' class. Interaction environment for the products.
85:  Class Client
86:      Private _abstractProductA As AbstractProductA
87:      Private _abstractProductB As AbstractProductB
88:      ' Constructor
89:      Public Sub New(factory As AbstractFactory)
90:          _abstractProductB = factory.CreateProductB()
91:          _abstractProductA = factory.CreateProductA()
92:      End Sub
93:      Public Sub Run()
94:          _abstractProductB.Interact(_abstractProductA)
95:      End Sub
96:  End Class
```

```
ProductB1 interacts with ProductA1
ProductB2 interacts with ProductA2
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects

- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/AbstractFactory.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

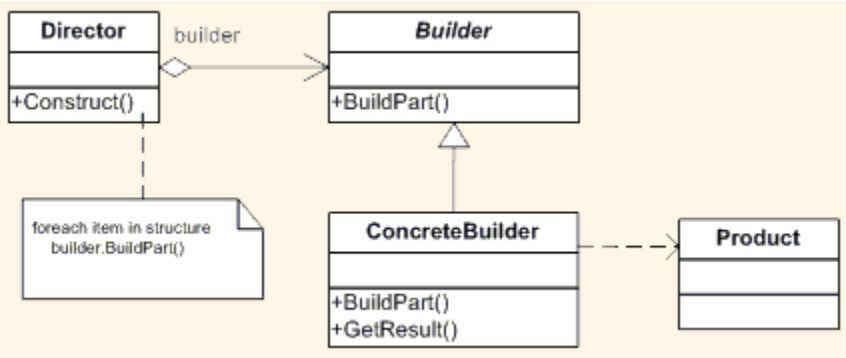
(NET) NET (2016)

Builder in VB.NET

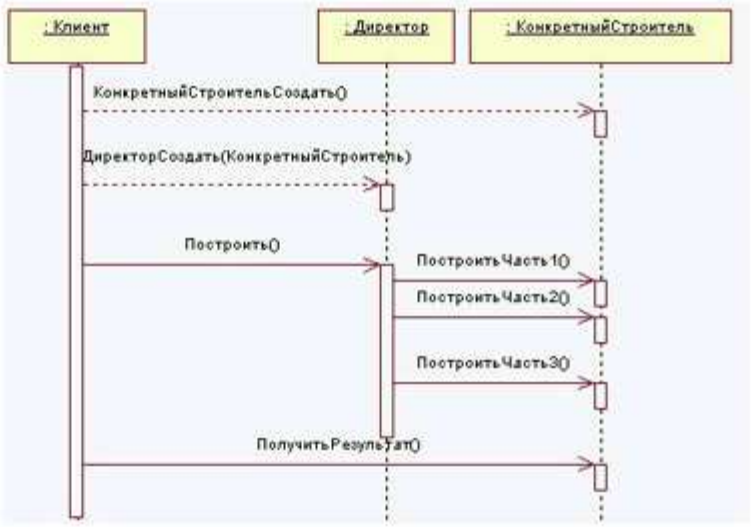
Separates object construction from its representation

назад

Separate the construction of a complex object from its representation so that the same construction process can create different representations. The construction process can create different object representations and provides a high level of control over the assembly of the objects.



3.3.5 Строитель (Builder) - GoF

Проблема	Отделить конструирование сложного объекта от его представления, так чтобы в результате одного и того же конструирования могли получаться различные представления. Алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой.
Решение	"Клиент" создает объект - распорядитель "Директор" и конфигурирует его объектом - "Строителем". "Директор" уведомляет "Строителя" о том, что нужно построить очередную часть "Продукта". "Строитель" обрабатывает запросы "Директора" и добавляет новые части к "Продукту", затем "Клиент" забирает "Продукт" у "Строителя".  
Преимущества	Объект "Строитель" предоставляет объекту "Директор" абстрактный интерфейс для конструирования "Продукта", за которым может скрыть представление и внутреннюю структуру продукта, и , кроме того, процесс сборки "продукта". Для изменения внутреннего представления "Продукта" достаточно определить новый вид "Строителя". Данный паттерн изолирует код, реализующий создание объекта и его представление.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Builder Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Builder.htm
3:      Public Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              ' Create director and builders
7:              Dim director As New Director()
8:              Dim b1 As Builder = New ConcreteBuilder1()
9:              Dim b2 As Builder = New ConcreteBuilder2()
10:             ' Construct two products
11:             director.Construct(b1)
12:             Dim p1 As Product = b1.GetResult()
13:             p1.Show()
14:             director.Construct(b2)
```

```
15:             Dim p2 As Product = b2.GetResult()
16:             p2.Show()
17:             ' Wait for user
18:             Console.ReadKey()
19:         End Sub
20:     End Class
21:
22:     ' The 'Director' class
23:     Class Director
24:         ' Builder uses a complex series of steps
25:         Public Sub Construct(builder As Builder)
26:             builder.BuildPartA()
27:             builder.BuildPartB()
28:         End Sub
29:     End Class
30:
31:     ' The 'Builder' abstract class
32:     MustInherit Class Builder
33:         Public MustOverride Sub BuildPartA()
34:         Public MustOverride Sub BuildPartB()
35:         Public MustOverride Function GetResult() As Product
36:     End Class
37:
38:     ' The 'ConcreteBuilder1' class
39:     Class ConcreteBuilder1
40:         Inherits Builder
41:         Private _product As New Product()
42:         Public Overrides Sub BuildPartA()
43:             _product.Add("PartA")
44:         End Sub
45:         Public Overrides Sub BuildPartB()
46:             _product.Add("PartB")
47:         End Sub
48:         Public Overrides Function GetResult() As Product
49:             Return _product
50:         End Function
51:     End Class
52:
53:     ' The 'ConcreteBuilder2' class
54:     Class ConcreteBuilder2
55:         Inherits Builder
56:         Private _product As New Product()
57:         Public Overrides Sub BuildPartA()
58:             _product.Add("PartX")
59:         End Sub
60:         Public Overrides Sub BuildPartB()
61:             _product.Add("PartY")
62:         End Sub
63:         Public Overrides Function GetResult() As Product
64:             Return _product
65:         End Function
66:     End Class
67:
68:     ' The 'Product' class
69:     Class Product
70:         Private _parts As New List(Of String)()
71:         Public Sub Add(part As String)
72:             _parts.Add(part)
73:         End Sub
74:         Public Sub Show()
75:             Console.WriteLine(vbLf & "Product Parts -------")
76:             For Each part As String In _parts
77:                 Console.WriteLine(part)
78:             Next
79:         End Sub
80:     End Class
```

```
Product Parts -------
PartA
PartB

Product Parts -------
PartX
PartY
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Builder.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

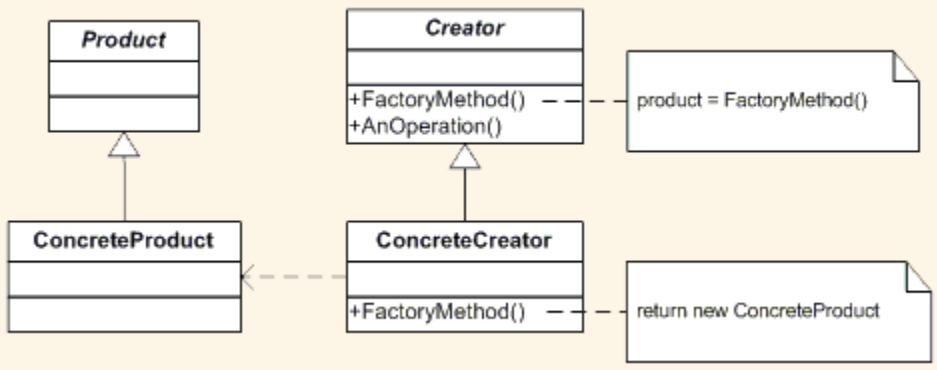
(NET) NET (2016)

Factory Method in VB.NET

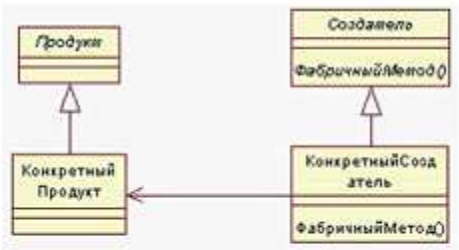
Creates an instance of several derived classes

назад

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses. The Abstract class may provide a default object, but each subclass can instantiate an extended version of the object.



3.3.6 (Фабричный метод) Factory Method или Виртуальный конструктор (Virtual Constructor) - GoF

Проблема	Определить интерфейс для создания объекта, но оставить подклассам решение о том, какой класс инстанцировать, то есть, делегировать инстанцирование подклассам.
Решение	Абстрактный класс "Создатель" объявляет ФабричныйМетод, возвращающий объект типа "Продукт" (абстрактный класс, определяющий интерфейс обьектов, создаваемых фабричным методом). "Создатель также может определить реализацию по умолчанию ФабричногоМетода, который возвращает "КонкретныйПродукт". "КонкретныйСоздатель" замещает ФабричныйМетод, возвращающий объект "КонкретныйПродукт". "Создатель" "полагается" на свои подклассы в определении ФабричногоМетода, возвращающего объект "КонкретныйПродукт". 
Преимущества	Избавляет проектировщика от необходимости встраивать в код зависящие от приложения классы.
Недостатки	Возникает дополнительный уровень подклассов.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```vb
' Factory Method Design Pattern.
' See description in http://www.vb-net.ru/ProgramTheory/FactoryMethod.htm
Class MainApp
    ' Entry point into console application.
    Public Shared Sub Main()
        ' An array of creators
        Dim creators As Creator() = New Creator(1) {}
        creators(0) = New ConcreteCreatorA()
        creators(1) = New ConcreteCreatorB()
        ' Iterate over creators and create products
        For Each creator As Creator In creators
            Dim product As Product1 = creator.FactoryMethod()
            Console.WriteLine("Created {0}", product.[GetType]().Name)
        Next
        ' Wait for user
        Console.ReadKey()
    End Sub
End Class

' The 'Product' abstract class
MustInherit Class Product1
End Class

' A 'ConcreteProduct' class
Class ConcreteProductA
    Inherits Product1
End Class

' A 'ConcreteProduct' class
Class ConcreteProductB
    Inherits Product1
End Class

' The 'Creator' abstract class
MustInherit Class Creator
    Public MustOverride Function FactoryMethod() As Product1
End Class

' A 'ConcreteCreator' class
Class ConcreteCreatorA
    Inherits Creator
    Public Overrides Function FactoryMethod() As Product1
        Return New ConcreteProductA()
    End Function
End Class

' A 'ConcreteCreator' class
Class ConcreteCreatorB
    Inherits Creator
    Public Overrides Function FactoryMethod() As Product1
        Return New ConcreteProductB()
    End Function
End Class
```

```
Created ConcreteProductA
Created ConcreteProductB
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem

- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/FactoryMethod.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

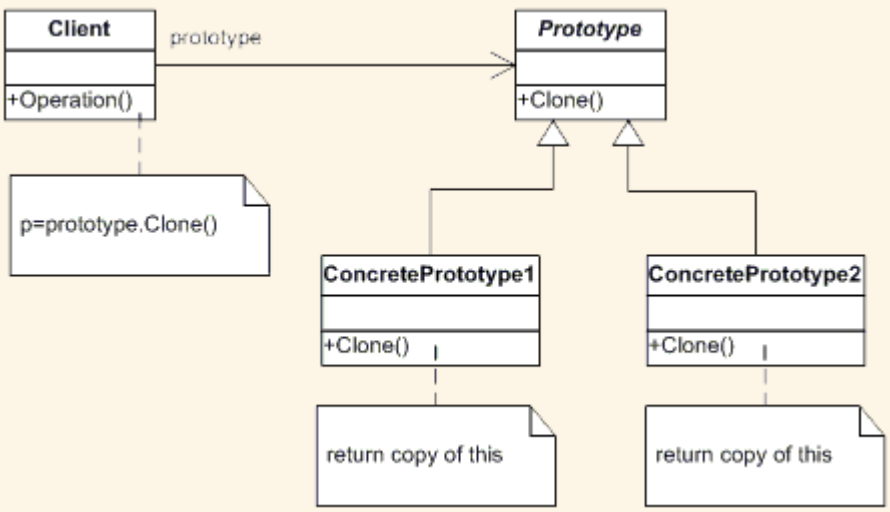
(NET) NET (2016)

Prototype in VB.NET

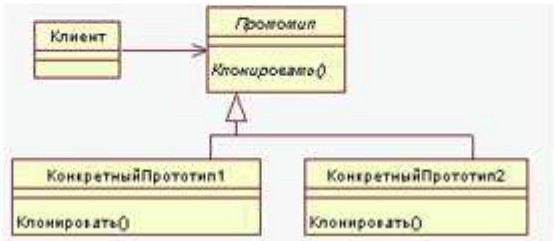
A fully initialized instance to be copied or cloned

назад

Specify the kind of objects to create using a prototypical instance, and create new objects by copying this prototype.



3.3.3 Прототип (Prototype) - GoF

Проблема	Система не должна зависеть от того, как в ней создаются, компонуются и представляются объекты.
Решение	Создавать новые объекты с помощью паттерна - прототипа. "Прототип" объявляет интерфейс для клонирования самого себя. "Клиент" создает новый объект, обращаясь к "Прототипу" с запросом клонировать "Прототип". 

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:  ' Prototype Design Pattern.
2:  ' See description in http://www.vb-net.ru/ProgramTheory/Prototype.htm
3:  Class MainApp
4:      ' Entry point into console application.
5:      Public Shared Sub Main()
6:          ' Create two instances and clone each
```

```
 7:          Dim p1 As New ConcretePrototype1("I")
 8:          Dim c1 As ConcretePrototype1 = DirectCast(p1.Clone(), ConcretePrototype1)
 9:          Console.WriteLine("Cloned: {0}", c1.Id)
10:          Dim p2 As New ConcretePrototype2("II")
11:          Dim c2 As ConcretePrototype2 = DirectCast(p2.Clone(), ConcretePrototype2)
12:          Console.WriteLine("Cloned: {0}", c2.Id)
13:          ' Wait for user
14:          Console.ReadKey()
15:      End Sub
16:  End Class
17:
18:  ' The 'Prototype' abstract class
19:  MustInherit Class Prototype
20:      Private _id As String
21:      ' Constructor
22:      Public Sub New(id As String)
23:          Me._id = id
24:      End Sub
25:      ' Gets id
26:      Public ReadOnly Property Id() As String
27:          Get
28:              Return _id
29:          End Get
30:      End Property
31:      Public MustOverride Function Clone() As Prototype
32:  End Class
33:
34:  ' A 'ConcretePrototype' class
35:  Class ConcretePrototype1
36:      Inherits Prototype
37:      ' Constructor
38:      Public Sub New(id As String)
39:          MyBase.New(id)
40:      End Sub
41:      ' Returns a shallow copy
42:      Public Overrides Function Clone() As Prototype
43:          Return DirectCast(Me.MemberwiseClone(), Prototype)
44:      End Function
45:  End Class
46:
47:  ' A 'ConcretePrototype' class
48:  Class ConcretePrototype2
49:      Inherits Prototype
50:      ' Constructor
51:      Public Sub New(id As String)
52:          MyBase.New(id)
53:      End Sub
54:      ' Returns a shallow copy
55:      Public Overrides Function Clone() As Prototype
56:          Return DirectCast(Me.MemberwiseClone(), Prototype)
57:      End Function
58:  End Class
```

```
Cloned: I
Cloned: II
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Prototype.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

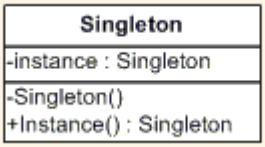
(NET) NET (2016)

Singleton in VB.NET

A class of which only a single instance can exist

назад

Ensure a class has only one instance and provide a global point of access to it.



3.3.2 Одиночка (Singleton) - GoF

Проблема	Какой специальный класс должен создавать "Абстрактную фабрику", см. 3.3.1 и как получить к ней доступ? Необходим лишь один экземпляр специального класса, различные объекты должны обращаться к этому экземпляру через единственную точку доступа.
Решение	Создать класс и определить статический метод класса, возвращающий этот единственный объект.
Рекомендации	Разумнее создавать именно статический экземпляр специального класса, а не объявить требуемые методы статическими, поскольку при использовании методов экземпляра можно применить механизм наследования и создавать подклассы. Статические методы в языках программирования не полиморфны и не допускают перекрытия в производных классах. Решение на основе создания экземпляра является более гибким, поскольку впоследствии может потребоваться уже не единственный экземпляр объекта, а несколько.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
 1:  ' Singleton Design Pattern.
 2:  ' See description in http://www.vb-net.ru/ProgramTheory/Singleton.htm
 3:  Class MainApp
 4:      ' Entry point into console application.
 5:      Public Shared Sub Main()
 6:          ' Constructor is protected -- cannot use new
 7:          Dim s1 As Singleton = Singleton.Instance()
 8:          Dim s2 As Singleton = Singleton.Instance()
 9:          ' Test for same instance
10:          If s1 Is s2 Then
11:              Console.WriteLine("Objects are the same instance")
12:          End If
13:          ' Wait for user
14:          Console.ReadKey()
15:      End Sub
16:  End Class
17:
18:  ' The 'Singleton' class
```

```
19:  Class Singleton
20:      Private Shared _instance As Singleton
21:      ' Constructor is 'protected'
22:      Protected Sub New()
23:      End Sub
24:
25:      Public Shared Function Instance() As Singleton
26:          ' Uses lazy initialization.
27:          ' Note: this is not thread safe.
28:          If _instance Is Nothing Then
29:              _instance = New Singleton()
30:          End If
31:          Return _instance
32:      End Function
33:  End Class
```

```
Objects are the same instance
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Singleton.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

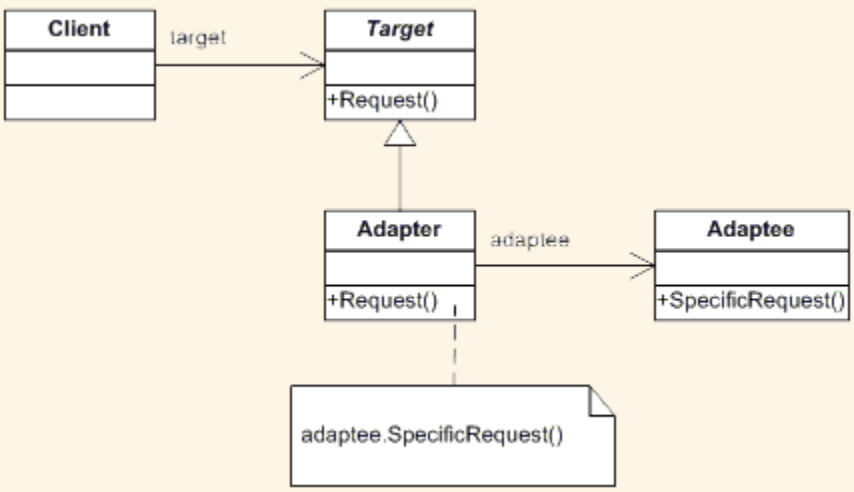
(NET) NET (2016)

Adapter in VB.NET

Match interfaces of different classes

назад

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces. These incompatible classes may come from different libraries or frameworks.



3.1.1 Адаптер (Adapter) - GoF

Проблема	Необходимо обеспечить взаимодействие несовместимых интерфейсов или как создать единый устойчивый интерфейс для нескольких компонентов с разными интерфейсами.
Решение	Конвертировать исходный интерфейс компонента к другому виду с помощью промежуточного объекта - адаптера, то есть, добавить специальный объект с общим интерфейсом в рамках данного приложения и перенаправить связи от внешних обьектов к этому объекту - адаптеру.
Пример	Соответствует примеру из описания паттерна "Полиморфизм", см. п. 3.2.15.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Adapter Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Adapter.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Private Shared Sub Main()
6:              ' Create adapter and place a request
7:              Dim target As Target = New Adapter()
8:              target.Request()
9:              ' Wait for user
10:             Console.ReadKey()
11:         End Sub
12:     End Class
```

```
13:
14:     ' The 'Target' class
15:     Class Target
16:         Public Overridable Sub Request()
17:             Console.WriteLine("Called Target Request()")
18:         End Sub
19:     End Class
20:
21:     ' The 'Adapter' class
22:     Class Adapter
23:         Inherits Target
24:         Private _adaptee As New Adaptee()
25:         Public Overrides Sub Request()
26:             ' Possibly do some other work
27:             '  and then call SpecificRequest
28:             _adaptee.SpecificRequest()
29:         End Sub
30:     End Class
31:
32:     ' The 'Adaptee' class
33:     Class Adaptee
34:         Public Sub SpecificRequest()
35:             Console.WriteLine("Called SpecificRequest()")
36:         End Sub
37:     End Class
```

```
Called SpecificRequest()
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Adapter.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

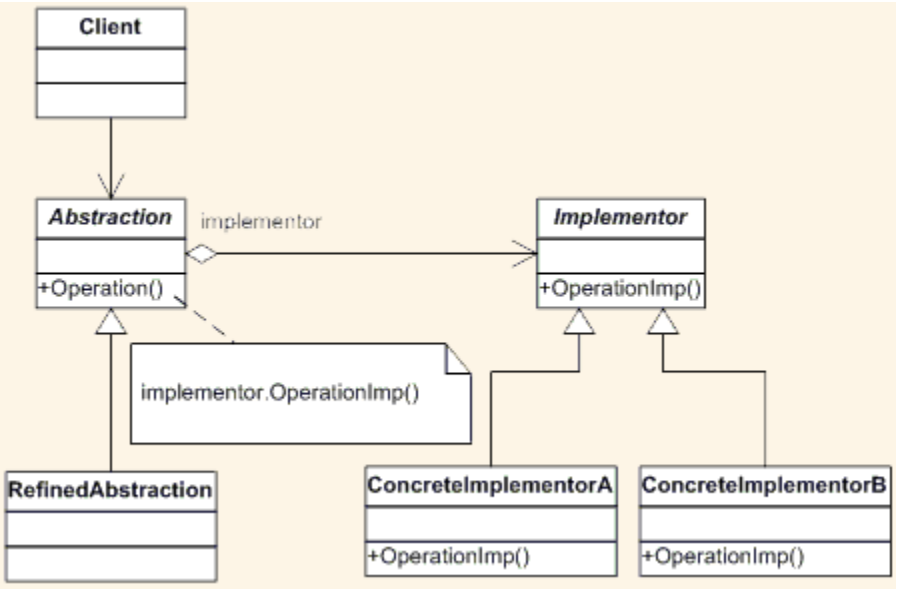
(NET) NET (2016)

Brige in VB.NET

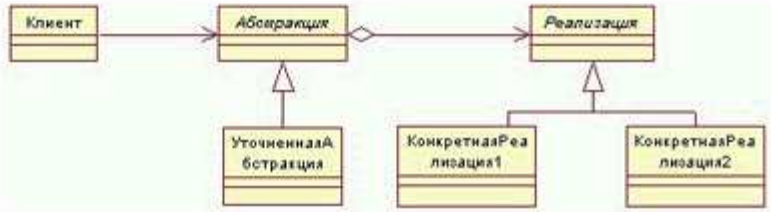
Separates an object’s interface from its implementation

назад

Decouple an abstraction from its implementation so that the two can vary independently. The implementation can evolve without changing clients which use the abstraction of the object.



3.1.6 Мост (Bridge), Handle (описатель) или Тело (Body) - GoF

Проблема	Требуется отделить абстракцию от реализации так, чтобы и то и другое можно было изменять независимо. При использовании наследования реализация жестко привязывается к абстракции, что затрудняет независимую модификацию.
Решение	Поместить абстракцию и реализацию в отдельные иерархии классов.
Рекомендации	Можно использовать если, например, реализацию необходимо выполнять во время реализации программы.
Пример	"Абстракция" определяет интерфейс "Абстракции" и хранит ссылку на объект "Реализация", "УточненнаяАбстракция" расширяет интерфейс, определенный "Абстракцией". "Реализация" определяет интерфейс для классов реализации, он не обязан точно соответствовать интерфейсу класса "Абстракция" - оба интерфейса могут быть совершенно различны. Обычно интерфейс класса "Реализация" предоставляет только примитивные операции, а класс "Абстракция" определяет операции более высокого уровня, базирующиеся на этих примитивных. "КонкретнаяРеализация" содержит конкретную реализацию класса "Реализация". Объект "Абстракция" перенаправляет своему объекту "Реализация" запросы "Клиента". 
Преимущества	Отделение реализации от интерфейса, то есть, "Реализацию" "Абстракции" можно конфигурировать во время выполнения. Кроме того, следует упомянуть, что разделение классов "Абстракция" и "Реализация" устраняет зависимости от реализации, устанавливаемые на этапе компиляции: чтобы изменить класс "Реализация" вовсе не обязательно перекомпилировать класс "Абстракция".

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Bridge Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Bridge.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              Dim ab As Abstraction = New RefinedAbstraction()
7:              ' Set implementation and call
8:              ab.Implementor = New ConcreteImplementorA()
9:              ab.Operation()
10:             ' Change implemention and call
11:             ab.Implementor = New ConcreteImplementorB()
12:             ab.Operation()
13:             ' Wait for user
14:             Console.ReadKey()
15:         End Sub
16:     End Class
17:
18:     ' The 'Abstraction' class
19:     Class Abstraction
20:         Protected _implementor As Implementor
21:         ' Property
22:         Public WriteOnly Property Implementor() As Implementor
23:             Set(value As Implementor)
24:                 _implementor = value
25:             End Set
26:         End Property
27:         Public Overridable Sub Operation()
28:             _implementor.Operation()
```

```
29:         End Sub
30:     End Class
31:     ' The 'Implementor' abstract class
32:     MustInherit Class Implementor
33:         Public MustOverride Sub Operation()
34:     End Class
35:
36:     ' The 'RefinedAbstraction' class
37:     Class RefinedAbstraction
38:         Inherits Abstraction
39:         Public Overrides Sub Operation()
40:             _implementor.Operation()
41:         End Sub
42:     End Class
43:
44:     ' The 'ConcreteImplementorA' class
45:     Class ConcreteImplementorA
46:         Inherits Implementor
47:         Public Overrides Sub Operation()
48:             Console.WriteLine("ConcreteImplementorA Operation")
49:         End Sub
50:     End Class
51:
52:     ' The 'ConcreteImplementorB' class
53:     Class ConcreteImplementorB
54:         Inherits Implementor
55:         Public Overrides Sub Operation()
56:             Console.WriteLine("ConcreteImplementorB Operation")
57:         End Sub
58:     End Class
```

```
ConcreteImplementorA Operation
ConcreteImplementorB Operation
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

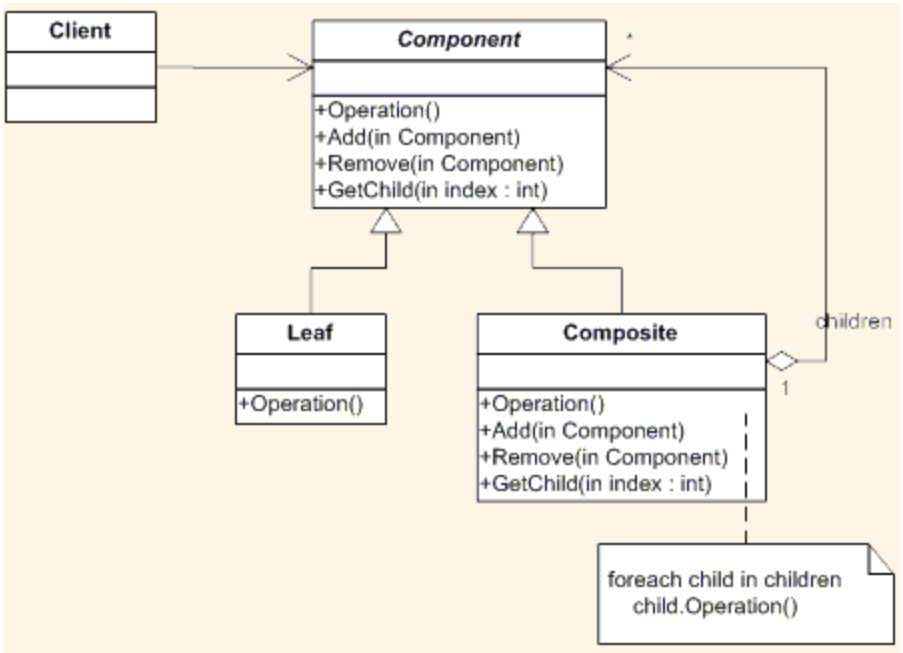
(NET) NET (2016)

Composite in VB.NET

A tree structure of simple and composite objects

назад

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. Composite pattern allows the creation of a tree structure in which individual nodes are accessed uniformly whether they are leaf nodes or branch (composite) nodes.



3.1.5 Компоновщик (Composite) - GoF

Проблема	Как обрабатывать группу или композицию структур обьектов одновременно?
Решение	Определить классы для композитных и атомарных обьектов таким образом, чтобы они реализовывали один и тот же интерфейс.
Пример	См. паттерн "Стратегия", 3.2.9, необходимо учесть несколько скидок различных видов (зависят от времени, типа покупателя, типом выбранного продукта. Как применять политику ценообразования? Вырабатывается стратегия приоритета скидок, объект "Продажа" не должен обладать информацией о применяемых скидках, но можно было бы применить стратегию расчета скидок. Создается новый класс "РасчетСкидкиАлгоритмКомпозит".

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Composite Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Composite.htm
3:      Class MainApp
4:          ' Entry point into console application.
```

```
 5:          Public Shared Sub Main()
 6:              ' Create a tree structure
 7:              Dim root As New Composite("root")
 8:              root.Add(New Leaf("Leaf A"))
 9:              root.Add(New Leaf("Leaf B"))
10:              Dim comp As New Composite("Composite X")
11:              comp.Add(New Leaf("Leaf XA"))
12:              comp.Add(New Leaf("Leaf XB"))
13:              root.Add(comp)
14:              root.Add(New Leaf("Leaf C"))
15:              ' Add and remove a leaf
16:              Dim leaf As New Leaf("Leaf D")
17:              root.Add(leaf)
18:              root.Remove(leaf)
19:              ' Recursively display tree
20:              root.Display(1)
21:              ' Wait for user
22:              Console.ReadKey()
23:          End Sub
24:      End Class
25:
26:      ' The 'Component' abstract class
27:      MustInherit Class Component
28:          Protected name As String
29:          ' Constructor
30:          Public Sub New(name As String)
31:              Me.name = name
32:          End Sub
33:          Public MustOverride Sub Add(c As Component)
34:          Public MustOverride Sub Remove(c As Component)
35:          Public MustOverride Sub Display(depth As Integer)
36:      End Class
37:
38:      ' The 'Composite' class
39:      Class Composite
40:          Inherits Component
41:          Private _children As New List(Of Component)()
42:          ' Constructor
43:          Public Sub New(name As String)
44:              MyBase.New(name)
45:          End Sub
46:          Public Overrides Sub Add(component As Component)
47:              _children.Add(component)
48:          End Sub
49:          Public Overrides Sub Remove(component As Component)
50:              _children.Remove(component)
51:          End Sub
52:          Public Overrides Sub Display(depth As Integer)
53:              Console.WriteLine(New [String]("-"c, depth) & name)
54:              ' Recursively display child nodes
55:              For Each component As Component In _children
56:                  component.Display(depth + 2)
57:              Next
58:          End Sub
59:      End Class
60:
61:      ' The 'Leaf' class
62:      Class Leaf
63:          Inherits Component
64:          ' Constructor
65:          Public Sub New(name As String)
66:              MyBase.New(name)
67:          End Sub
68:          Public Overrides Sub Add(c As Component)
69:              Console.WriteLine("Cannot add to a leaf")
70:          End Sub
71:          Public Overrides Sub Remove(c As Component)
72:              Console.WriteLine("Cannot remove from a leaf")
73:          End Sub
74:          Public Overrides Sub Display(depth As Integer)
75:              Console.WriteLine(New [String]("-"c, depth) & name)
76:          End Sub
77:      End Class
```

```
-root
---Leaf A
---Leaf B
---Composite X
-----Leaf XA
-----Leaf XB
---Leaf C
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Composite.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

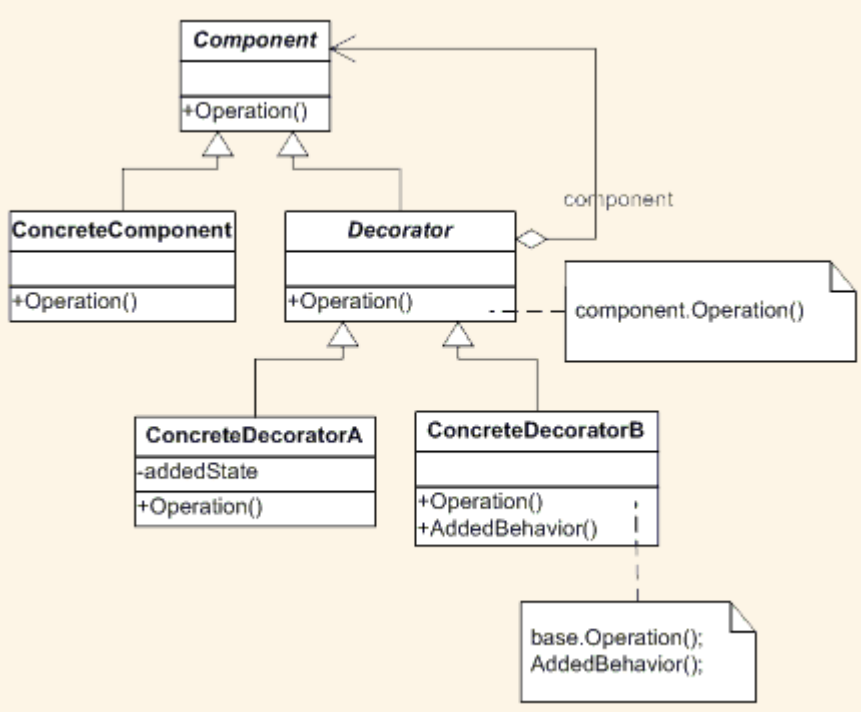
(NET) NET (2016)

Decorator in VB.NET

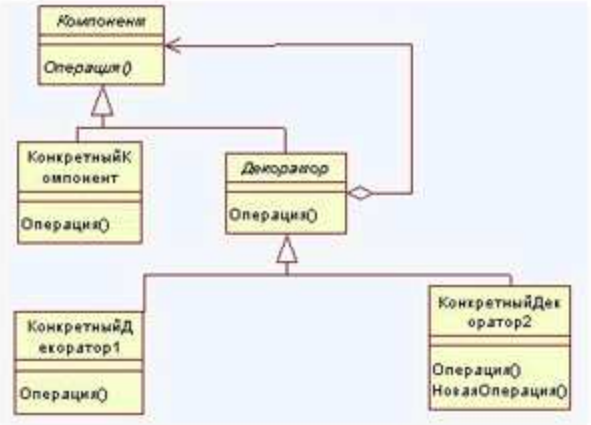
Add responsibilities to objects dynamically

назад

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. Decorator pattern dynamically adds extra functionality to an existing object.



3.1.2 Декоратор (Decorator) или Оболочка (Wrapper) - GoF

Проблема	Возложить дополнительные обязанности (прозрачные для клиентов) на отдельный объект, а не на класс в целом.
Рекомендации	Применение нескольких "Декораторов" к одному "Компоненту" позволяет произвольным образом сочетать обязанности, например, одно свойство можно добавить дважды.
Решение	Динамически добавить объекту новые обязанности не прибегая при этом к порождению подклассов (наследованию). "Компонент"определяет интерфейс для обьектов, на которые могут быть динамически возложены дополнительные обязанности, "КонкретныйКомпонент" определяет объект, на который возлагаются дополнительные обязанности, "Декоратор" - хранит ссылку на объект "Компонент" и определяет интерфейс, соответствующий интерфейсу "Компонента". "КонкретныйДекоратор" возлагает дополнительные обязанности на компонент. "Декоратор" переадресует запросы объекту "Компонент". 
Преимущества	Большая гибкость, чем у статического наследования: можно добавлять и удалять обязанности во время выполнения программы в то время как при использовании наследования надо было бы создавать новый класс для каждой дополнительной обязанности. Данный паттерн позволяет избежать перегруженных методами классов на верхних уровнях иерархии - новые обязанности можно добавлять по мере необходимости.
Недостатки	"Декоратор" и его "Компонент" не идентичны, и, кроме того, получается что система состоит из большого числа мелких обьектов, которые похожи друг на друга и различаются только способом взаимосвязи а не классом и не значениями своих внутренних переменных - такая система сложна в изучении и отладке.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
 1:      ' Decorator Design Pattern.
 2:      ' See description in http://www.vb-net.ru/ProgramTheory/Decorator.htm
 3:      Class MainApp
 4:          ' Entry point into console application.
 5:          Public Shared Sub Main()
 6:              ' Create ConcreteComponent and two Decorators
 7:              Dim c As New ConcreteComponent()
 8:              Dim d1 As New ConcreteDecoratorA()
 9:              Dim d2 As New ConcreteDecoratorB()
10:              ' Link decorators
11:              d1.SetComponent(c)
12:              d2.SetComponent(d1)
13:              d2.Operation()
14:              ' Wait for user
15:              Console.ReadKey()
16:          End Sub
```

```
17:     End Class
18:
19:     ' The 'Component' abstract class
20:     MustInherit Class Component
21:         Public MustOverride Sub Operation()
22:     End Class
23:
24:     ' The 'ConcreteComponent' class
25:     Class ConcreteComponent
26:         Inherits Component
27:         Public Overrides Sub Operation()
28:             Console.WriteLine("ConcreteComponent.Operation()")
29:         End Sub
30:     End Class
31:
32:     ' The 'Decorator' abstract class
33:     MustInherit Class Decorator
34:         Inherits Component
35:         Protected component As Component
36:         Public Sub SetComponent(component As Component)
37:             Me.component = component
38:         End Sub
39:         Public Overrides Sub Operation()
40:             If component IsNot Nothing Then
41:                 component.Operation()
42:             End If
43:         End Sub
44:     End Class
45:
46:     ' The 'ConcreteDecoratorA' class
47:     Class ConcreteDecoratorA
48:         Inherits Decorator
49:         Public Overrides Sub Operation()
50:             MyBase.Operation()
51:             Console.WriteLine("ConcreteDecoratorA.Operation()")
52:         End Sub
53:     End Class
54:
55:     ' The 'ConcreteDecoratorB' class
56:     Class ConcreteDecoratorB
57:         Inherits Decorator
58:         Public Overrides Sub Operation()
59:             MyBase.Operation()
60:             AddedBehavior()
61:             Console.WriteLine("ConcreteDecoratorB.Operation()")
62:         End Sub
63:         Private Sub AddedBehavior()
64:         End Sub
65:     End Class
```

```
ConcreteComponent.Operation()
ConcreteDecoratorA.Operation()
ConcreteDecoratorB.Operation()
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects

- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Decorator.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

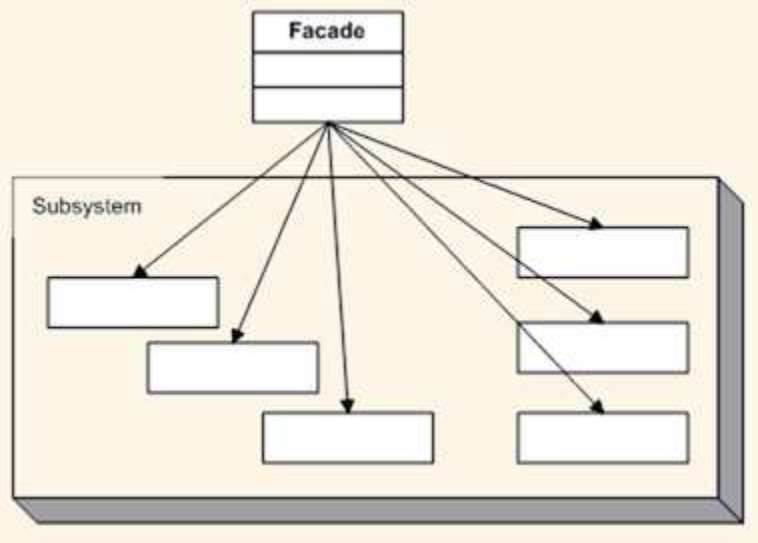
(NET) NET (2016)

Facade in VB.NET

A single class that represents an entire subsystem

назад

Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. Facade pattern provides a simplified and uniform interface to a large subsystem of classes.



3.1.10 Фасад (Facade) - GoF

Проблема	Как обеспечить унифицированный интерфейс с набором разрозненных реализаций или интерфейсов, например, с подсистемой, если нежелательно высокое связывание с этой подсистемой или реализация подсистемы может измениться?
Решение	Определить одну точку взаимодействия с подсистемой - фасадный объект, обеспечивающий общий интерфейс с подсистемой и возложить на него обязанность по взаимодействию с ее компонентами. Фасад - это внешний объект, обеспечивающий единственную точку входа для служб подсистемы. Реализация других компонентов подсистемы закрыта и не видна внешним компонентам. Фасадный объект обеспечивает реализацию паттерна "Устойчивый к изменениям" с точки зрения защиты от изменений в реализации подсистемы., см. п. 3.1.9.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Facade Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Facade.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              Dim facade As New Facade()
7:              facade.MethodA()
8:              facade.MethodB()
```

```
 9:              ' Wait for user
10:              Console.ReadKey()
11:          End Sub
12:      End Class
13:
14:      ' The 'Subsystem ClassA' class
15:      Class SubSystemOne
16:          Public Sub MethodOne()
17:              Console.WriteLine(" SubSystemOne Method")
18:          End Sub
19:      End Class
20:
21:      ' The 'Subsystem ClassB' class
22:      Class SubSystemTwo
23:          Public Sub MethodTwo()
24:              Console.WriteLine(" SubSystemTwo Method")
25:          End Sub
26:      End Class
27:
28:      ' The 'Subsystem ClassC' class
29:      Class SubSystemThree
30:          Public Sub MethodThree()
31:              Console.WriteLine(" SubSystemThree Method")
32:          End Sub
33:      End Class
34:
35:      ' The 'Subsystem ClassD' class
36:      Class SubSystemFour
37:          Public Sub MethodFour()
38:              Console.WriteLine(" SubSystemFour Method")
39:          End Sub
40:      End Class
41:
42:      ' The 'Facade' class
43:      Class Facade
44:          Private _one As SubSystemOne
45:          Private _two As SubSystemTwo
46:          Private _three As SubSystemThree
47:          Private _four As SubSystemFour
48:          Public Sub New()
49:              _one = New SubSystemOne()
50:              _two = New SubSystemTwo()
51:              _three = New SubSystemThree()
52:              _four = New SubSystemFour()
53:          End Sub
54:          Public Sub MethodA()
55:              Console.WriteLine(vbLf & "MethodA() ---- ")
56:              _one.MethodOne()
57:              _two.MethodTwo()
58:              _four.MethodFour()
59:          End Sub
60:          Public Sub MethodB()
61:              Console.WriteLine(vbLf & "MethodB() ---- ")
62:              _two.MethodTwo()
63:              _three.MethodThree()
64:          End Sub
65:      End Class
```

```
MethodA() ----
SubSystemOne Method
SubSystemTwo Method
SubSystemFour Method

MethodB() ----
SubSystemTwo Method
SubSystemThree Method
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Facade.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

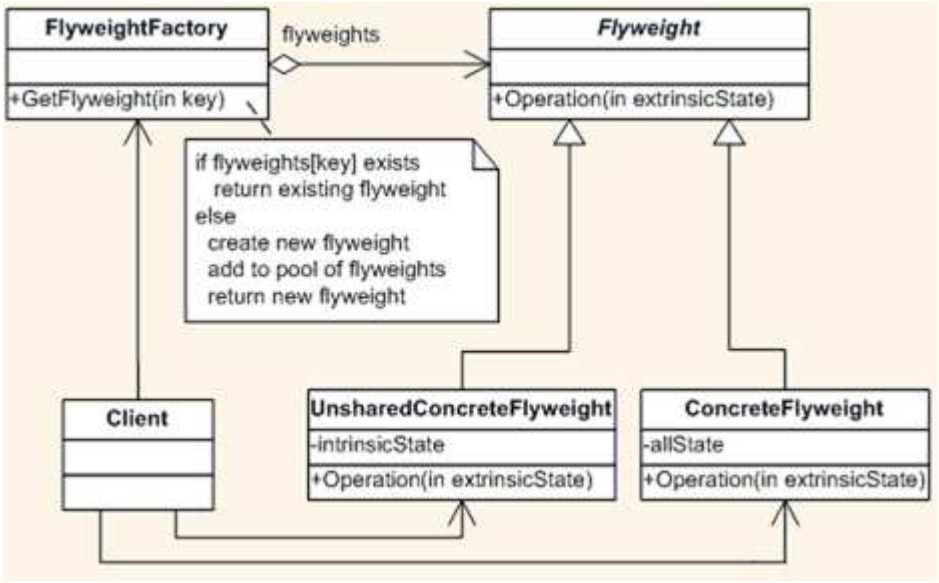
(NET) NET (2016)

Flyweight in VB.NET

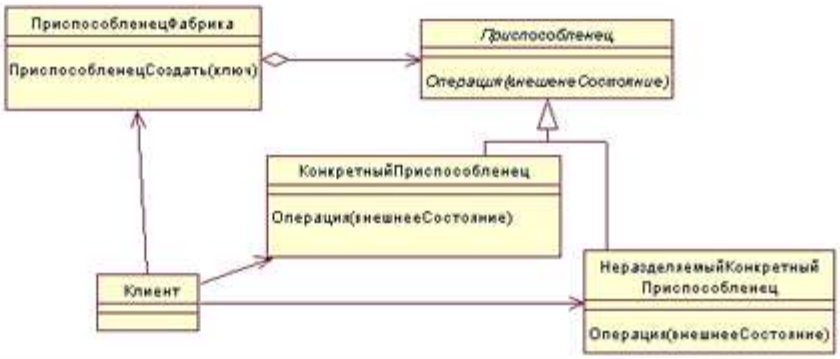
A fine-grained instance used for efficient sharing

назад

Use sharing to support large numbers of fine-grained objects efficiently. Flyweight pattern with a relatively small number of objects is shared many times by different clients.



3.1.8 Приспособленец (Flyweight) - GoF

Проблема	Необходимо обеспечить поддержку множества мелких объектов.
Рекомендации	Приспособленцы моделируют сущности, число которых слишком велико для представления объектами. Имеет смысл использовать данный паттерн если одновременно выполняются следующие условия: • в приложении используется большое число обьектов, из-за этого расходы на хранение высоки, • большую часть состояния обьектов можно вынести вовне, • многие группы обьектов можно заменить относительно небольшим количеством обьектов, поскольку состояния обьектов вынесены вовне.
Решение	Создать разделяемый объект, который можно использовать одновременно в нескольких контекстах, причем, в каждом контексте он выглядит как независимый объект (неотличим от экземпляра, который не разделяется). "Приспособленец" объявляет интерфейс, с помощью которого приспособленцы могут получить внешнее состояние или как-то воздействовать на него, "КонкретныйПриспособленец" реализует интерфейс класса "Приспособленец" и добавляет при необходимости внутреннее состояние. Внутреннее состояние хранится в объекте "КонкретныйПриспособленец", в то время как внешнее состояние хранится или вычисляется "Клиентами" ("Клиент" передает его "Приспособленцу" при вызове операций). Объект класса "КонкретныйПриспособленец" должен быть разделяемым. Любое сохраняемое им состояние должно быть внутренним, то есть независимым от контекста, "ПриспособленецФабрика" - создает объекты - "Приспособленцы" (или предоставляет существующий экземпляр) и управляет ими. "НеразделяемыйКонкретныйПриспособленец" - не все подклассы "Приспособленца" обязательно должны быть разделяемыми. "Клиент" - хранит ссылки на одного или нескольких "Приспособленцев", вычисляет и хранит внешнее состояние "Приспособленцев". 
Преимущества	Вследствие уменьшения общего числа экземпляров и вынесения состояния экономится память.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Flyweight Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Flyweight.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              ' Arbitrary extrinsic state
7:              Dim extrinsicstate As Integer = 22
8:              Dim factory As New FlyweightFactory()
9:              ' Work with different flyweight instances
```

```
10:             Dim fx As Flyweight = factory.GetFlyweight("X")
11:             extrinsicstate = extrinsicstate - 1
12:             fx.Operation(extrinsicstate)
13:             Dim fy As Flyweight = factory.GetFlyweight("Y")
14:             extrinsicstate = extrinsicstate - 1
15:             fy.Operation(extrinsicstate)
16:             Dim fz As Flyweight = factory.GetFlyweight("Z")
17:             extrinsicstate = extrinsicstate - 1
18:             fz.Operation(extrinsicstate)
19:             Dim fu As New UnsharedConcreteFlyweight()
20:             extrinsicstate = extrinsicstate - 1
21:             fu.Operation(extrinsicstate)
22:             ' Wait for user
23:             Console.ReadKey()
24:         End Sub
25:     End Class
26:
27:     ' The 'FlyweightFactory' class
28:     Class FlyweightFactory
29:         Private flyweights As New Hashtable()
30:         ' Constructor
31:         Public Sub New()
32:             flyweights.Add("X", New ConcreteFlyweight())
33:             flyweights.Add("Y", New ConcreteFlyweight())
34:             flyweights.Add("Z", New ConcreteFlyweight())
35:         End Sub
36:         Public Function GetFlyweight(key As String) As Flyweight
37:             Return DirectCast(flyweights(key), Flyweight)
38:         End Function
39:     End Class
40:
41:     ' The 'Flyweight' abstract class
42:     MustInherit Class Flyweight
43:         Public MustOverride Sub Operation(extrinsicstate As Integer)
44:     End Class
45:
46:     ' The 'ConcreteFlyweight' class
47:     Class ConcreteFlyweight
48:         Inherits Flyweight
49:         Public Overrides Sub Operation(extrinsicstate As Integer)
50:             Console.WriteLine("ConcreteFlyweight: " & extrinsicstate.ToString)
51:         End Sub
52:     End Class
53:
54:     ' The 'UnsharedConcreteFlyweight' class
55:     Class UnsharedConcreteFlyweight
56:         Inherits Flyweight
57:         Public Overrides Sub Operation(extrinsicstate As Integer)
58:             Console.WriteLine("UnsharedConcreteFlyweight: " & extrinsicstate.ToString)
59:         End Sub
60:     End Class
```

```
ConcreteFlyweight: 21
ConcreteFlyweight: 20
ConcreteFlyweight: 19
UnsharedConcreteFlyweight: 18
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing

- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Flyweight.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

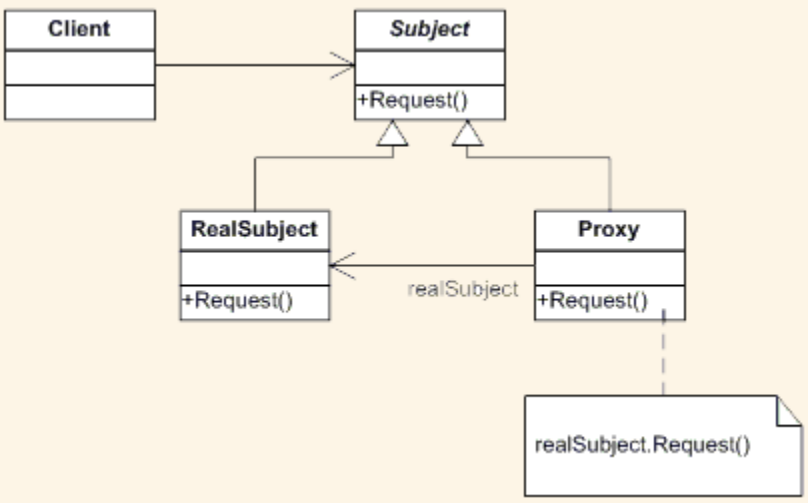
(NET) NET (2016)

Proxy in VB.NET

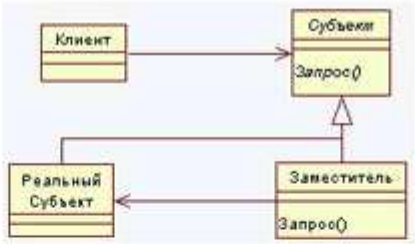
An object representing another object

назад

Provide a surrogate or placeholder for another object to control access to it. Proxy pattern provides a representative object (proxy) that controls access to another similar object.



3.1.3 Заместитель (Proxy) или Суррогат (Surrogate) - GoF

Проблема	Необходимо управлять доступом к объекту, так чтобы создавать громоздкие объекты "по требованию".
Решение	Создать суррогат громоздкого объекта. "Заместитель" хранит ссылку, которая позволяет заместителю обратиться к реальному субъекту (объект класса "Заместитель" может обращаться к объекту класса "Субъект", если интерфейсы "РеальногоСубъекта" и "Субъекта" одинаковы). Поскольку интерфейс "РеальногоСубъекта" идентичен интерфейсу "Субъекта", так, что "Заместителя" можно подставить вместо "РеальногоСубъекта", контролирует доступ к "РеальномуСубъекту", может отвечать за создание или удаление "РеальногоСубъекта". "Субъект" определяет общий для "РеальногоСубъекта" и "Заместителя" интерфейс, так, что "Заместитель" может быть использован везде, где ожидается "РеальныйСубъект". При необходимости запросы могут быть переадресованы "Заместителем" "РеальномуСубъекту".  "Заместитель" может иметь и другие обязанности, а именно: • удаленный "Заместитель" может отвечать за кодирование запроса и его аргументов и отправку закодированного запроса реальному "Субъекту", • виртуальный "Заместитель" может кэшировать дополнительную информацию о реальном "Субъекте", чтобы отложить его создание, • защищающий "Заместитель" может проверять, имеет ли вызывающий объект необходимые для выполнения запроса права.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Proxy Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Proxy.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              ' Create proxy and request a service
7:              Dim proxy As New Proxy()
8:              proxy.Request()
9:              ' Wait for user
10:             Console.ReadKey()
11:         End Sub
12:     End Class
13:     ' The 'Subject' abstract class
14:     MustInherit Class Subject
15:         Public MustOverride Sub Request()
16:     End Class
17:
18:     ' The 'RealSubject' class
19:     Class RealSubject
20:         Inherits Subject
21:         Public Overrides Sub Request()
22:             Console.WriteLine("Called RealSubject.Request()")
23:         End Sub
24:     End Class
25:
26:     ' The 'Proxy' class
27:     Class Proxy
28:         Inherits Subject
29:         Private _realSubject As RealSubject
```

```
30:          Public Overrides Sub Request()
31:              ' Use 'lazy initialization'
32:              If _realSubject Is Nothing Then
33:                  _realSubject = New RealSubject()
34:              End If
35:              _realSubject.Request()
36:          End Sub
37:      End Class
```

```
Called RealSubject.Request()
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Proxy.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> <

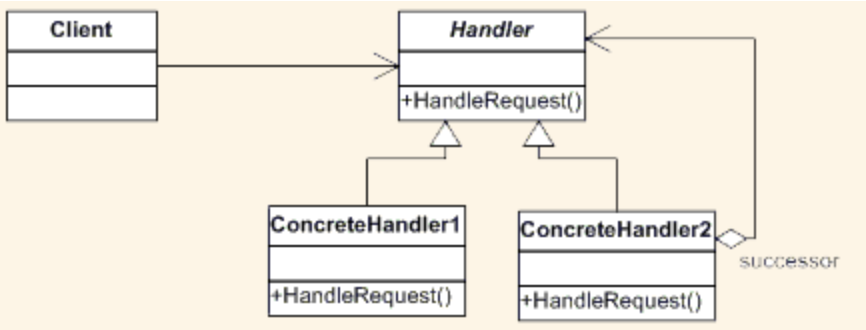
(NET) NET (2016)

Chain of Resp. in VB.NET

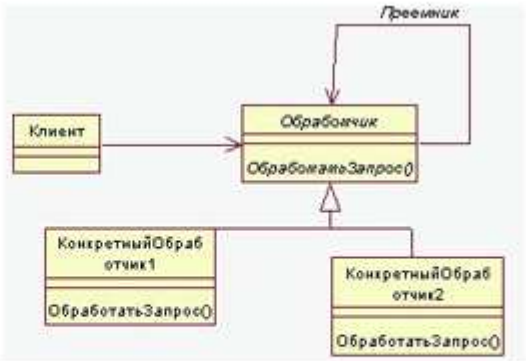
A way of passing a request between a chain of objects

назад

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it. Chain of Responsibility pattern with several linked objects (the Chain) are offered the opportunity to respond to a request or hand it off to the object next in line.



3.2.11 Цепочка обязанностей (Chain of Responsibility) - GoF

Проблема	Запрос должен быть обработан несколькими объектами.
Рекомендации	Логично использовать данный паттерн, если имеется более одного объекта, способного обработать запрос и обработчик заранее неизвестен (и должен быть найден автоматически) или если весь набор обьектов, которые способны обработать запрос, должен задаваться автоматически.
Решение	Связать объекты - получатели запроса в цепочку и передать запрос вдоль этой цепочки, пока он не будет обработан. "Обработчик" определяет интерфейс для обработки запросов, и, возможно, реализует связь с преемником, "КонкретныйОбработчик" обрабатывает запрос, за который отвечает, имеет доступ к своему преемнику ("КонкретныйОбработчик" направляет запрос к своему преемнику, если не может обработать запрос сам. 
Преимущества	Ослабляется связанность (объект не обязан "знать", кто именно обработает его запрос).
Недостатки	Нет гарантий, что запрос будет обработан, поскольку он не имеет явного получателя.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Chain of Responsibility Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/ChainOfResp.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              ' Setup Chain of Responsibility
7:              Dim h1 As Handler = New ConcreteHandler1()
8:              Dim h2 As Handler = New ConcreteHandler2()
9:              Dim h3 As Handler = New ConcreteHandler3()
10:             h1.SetSuccessor(h2)
11:             h2.SetSuccessor(h3)
12:             ' Generate and process request
13:             Dim requests As Integer() = {2, 5, 14, 22, 18, 3, 27, 20}
14:             For Each request As Integer In requests
15:                 h1.HandleRequest(request)
16:             Next
17:             ' Wait for user
18:             Console.ReadKey()
19:         End Sub
20:     End Class
21:
22:     ' The 'Handler' abstract class
23:     MustInherit Class Handler
24:         Protected successor As Handler
25:         Public Sub SetSuccessor(successor As Handler)
26:             Me.successor = successor
27:         End Sub
28:         Public MustOverride Sub HandleRequest(request As Integer)
29:     End Class
30:
31:     ' The 'ConcreteHandler1' class
32:     Class ConcreteHandler1
```

```
33:          Inherits Handler
34:          Public Overrides Sub HandleRequest(request As Integer)
35:              If request >= 0 AndAlso request < 10 Then
36:                  Console.WriteLine("{0} handled request {1}", Me.GetType.Name, request)
37:              ElseIf successor IsNot Nothing Then
38:                  successor.HandleRequest(request)
39:              End If
40:          End Sub
41:      End Class
42:
43:      ' The 'ConcreteHandler2' class
44:      Class ConcreteHandler2
45:          Inherits Handler
46:          Public Overrides Sub HandleRequest(request As Integer)
47:              If request >= 10 AndAlso request < 20 Then
48:                  Console.WriteLine("{0} handled request {1}", Me.GetType.Name, request)
49:              ElseIf successor IsNot Nothing Then
50:                  successor.HandleRequest(request)
51:              End If
52:          End Sub
53:      End Class
54:
55:      ' The 'ConcreteHandler3' class
56:      Class ConcreteHandler3
57:          Inherits Handler
58:          Public Overrides Sub HandleRequest(request As Integer)
59:              If request >= 20 AndAlso request < 30 Then
60:                  Console.WriteLine("{0} handled request {1}", Me.GetType.Name, request)
61:              ElseIf successor IsNot Nothing Then
62:                  successor.HandleRequest(request)
63:              End If
64:          End Sub
65:      End Class
```

```
ConcreteHandler1 handled request 2
ConcreteHandler1 handled request 5
ConcreteHandler2 handled request 14
ConcreteHandler3 handled request 22
ConcreteHandler2 handled request 18
ConcreteHandler1 handled request 3
ConcreteHandler3 handled request 27
ConcreteHandler3 handled request 20
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass

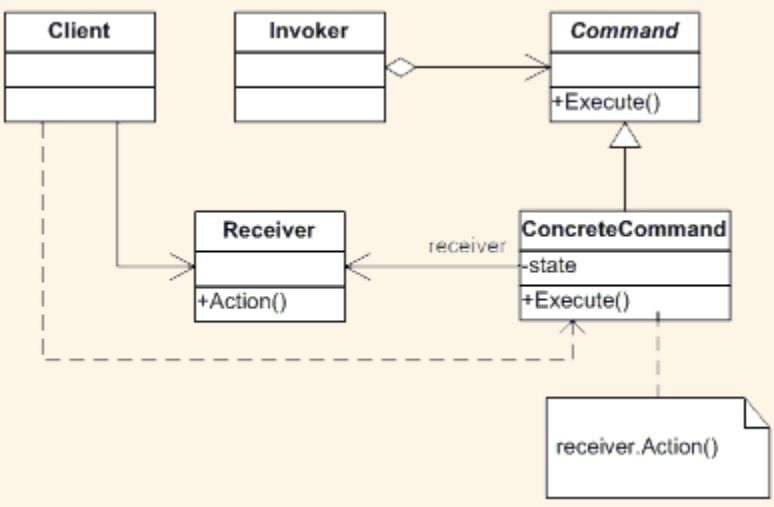
(NET) NET (2016)

Command in VB.NET

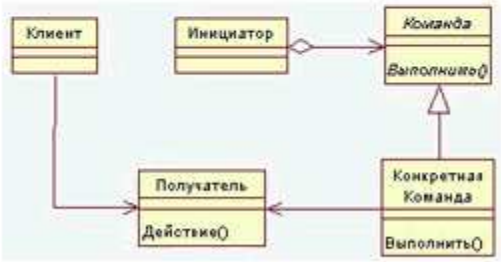
Encapsulate a command request as an object

назад

Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations. Command pattern which stores requests as objects allow clients to execute or playback the requests.



3.2.3 Команда (Command), Действие (Action) или Транзакция (Транзакция) - GoF

Проблема	Необходимо послать объекту запрос, не зная о том, выполнение какой операции запрошено и кто будет получателем.
Решение	Инкапсулировать запрос как объект. "Клиент" создает объект "КонкретнаяКоманда", который вызывает операции получателя для выполнения запроса, "Инициатор" отправляет запрос, выполоняя операцию "Команды" Выполнить(). "Команда" объявляет интерфейс для выполнения операции, "КонкретнаяКоманда" определяет связь между объектом "Получатель" и операцией Действие(), и, кроме того, реализует операцию Выполнить() путем вызова соответствующих операций объекта "Получатель". "Клиент" создает экземпляр класса "КонкретнаяКоманда" и устанавливает его получателя, "Инициатор" обращается к команде для выполнения запроса, "Получатель" (любой класс) располагает информацией о способах выполнения операций, необходимых для выполнения запроса. 
Преимущества	Паттерн "Команда" разрывает связь между объектом, инициирующим операции, и объектом, имеющим информацию о том, как ее выполнить, кроме того создается объект "Команда", который можно расширять и манипулировать им как объектом.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Command Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Command.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              ' Create receiver, command, and invoker
7:              Dim receiver As New Receiver()
8:              Dim command As Command = New ConcreteCommand(receiver)
9:              Dim invoker As New Invoker()
10:             ' Set and execute command
11:             invoker.SetCommand(command)
12:             invoker.ExecuteCommand()
13:             ' Wait for user
14:             Console.ReadKey()
15:         End Sub
16:     End Class
17:
18:     ' The 'Command' abstract class
19:     MustInherit Class Command
20:         Protected receiver As Receiver
21:         ' Constructor
22:         Public Sub New(receiver As Receiver)
23:             Me.receiver = receiver
24:         End Sub
25:         Public MustOverride Sub Execute()
26:     End Class
27:
28:     ' The 'ConcreteCommand' class
29:     Class ConcreteCommand
30:         Inherits Command
31:         ' Constructor
32:         Public Sub New(receiver As Receiver)
33:             MyBase.New(receiver)
34:         End Sub
```

```
35:          Public Overrides Sub Execute()
36:              receiver.Action()
37:          End Sub
38:      End Class
39:
40:      ' The 'Receiver' class
41:      Class Receiver
42:          Public Sub Action()
43:              Console.WriteLine("Called Receiver.Action()")
44:          End Sub
45:      End Class
46:
47:      ' The 'Invoker' class
48:      Class Invoker
49:          Private _command As Command
50:          Public Sub SetCommand(command As Command)
51:              Me._command = command
52:          End Sub
53:          Public Sub ExecuteCommand()
54:              _command.Execute()
55:          End Sub
56:      End Class
```

```
Called Receiver.Action()
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Command.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

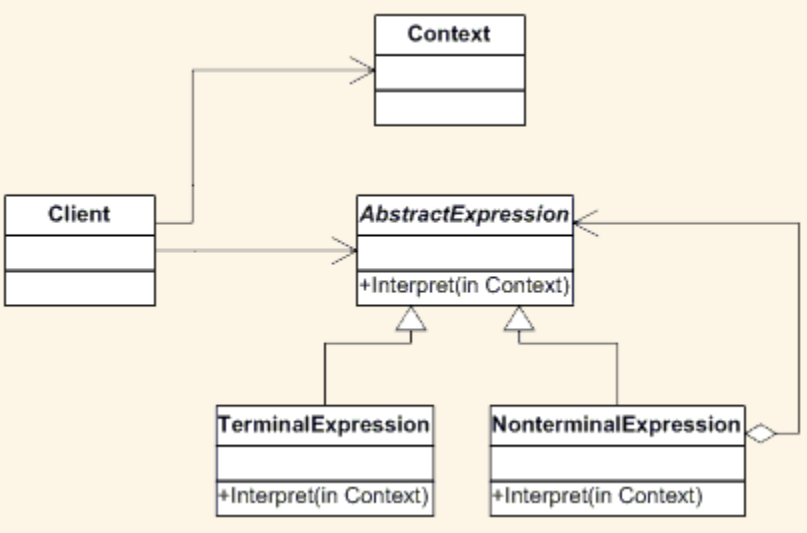
(NET) NET (2016)

Interpreter in VB.NET

A way to include language elements in a program

назад

Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language. Interpreter patterns, which using a defined grammer, provides the interpreter that processes parsed statements.



3.2.1 Интерпретатор (Interpreter) - GoF

Проблема	Имеется часто встречающаяся, подверженная изменениям задача.
Решение	Создать интерпретатор, который решает данную задачу.
Пример	Задача поиска строк по образцу может быть решена посредством создания интерпретатора, определяющего грамматику языка. "Клиент" строит предложение в виде абстрактного синтаксического дерева, в узлах которого находятся объекты классов "НетерминальноеВыражение" и "ТерминальноеВыражение" (рекурсивное), затем "Клиент" инициализирует контекст и вызывает операцию Разобрать(Контекст). На каждом узле типа "НетерминальноеВыражение" определяется операция Разобрать для каждого подвыражения. Для класса "ТерминальноеВыражение" операция Разобрать определяет базу рекурсии. "АбстрактноеВыражение" определяет абстрактную операцию Разобрать, общую для всех узлов в абстрактном синтаксическом дереве. "Контекст" содержит информацию, глобальную по отношению к интерпретатору. 
Преимущества	Грамматику становится легко расширять и изменять, реализации классов, описывающих узлы абстрактного синтаксического дерева похожи (легко кодируются). Можно легко изменять способ вычисления выражений.
Недостатки	Сопровождение грамматики с большим числом правил затруднительно.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Interpreter Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Interpreter.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              Dim context As New Context()
7:              ' Usually a tree
8:              Dim list As New ArrayList()
9:              ' Populate 'abstract syntax tree'
10:             list.Add(New TerminalExpression())
11:             list.Add(New NonterminalExpression())
12:             list.Add(New TerminalExpression())
13:             list.Add(New TerminalExpression())
14:             ' Interpret
15:             For Each exp As AbstractExpression In list
16:                 exp.Interpret(context)
17:             Next
18:             ' Wait for user
19:             Console.ReadKey()
20:         End Sub
21:     End Class
22:
23:     ' The 'Context' class
24:     Class Context
25:     End Class
26:
27:     ' The 'AbstractExpression' abstract class
28:     MustInherit Class AbstractExpression
29:         Public MustOverride Sub Interpret(context As Context)
30:     End Class
```

```
31:
32:     ' The 'TerminalExpression' class
33:     Class TerminalExpression
34:         Inherits AbstractExpression
35:         Public Overrides Sub Interpret(context As Context)
36:             Console.WriteLine("Called Terminal.Interpret()")
37:         End Sub
38:     End Class
39:
40:     ' The 'NonterminalExpression' class
41:     Class NonterminalExpression
42:         Inherits AbstractExpression
43:         Public Overrides Sub Interpret(context As Context)
44:             Console.WriteLine("Called Nonterminal.Interpret()")
45:         End Sub
46:     End Class
```

```
Called Terminal.Interpret()
Called Nonterminal.Interpret()
Called Terminal.Interpret()
Called Terminal.Interpret()
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Interpreter.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

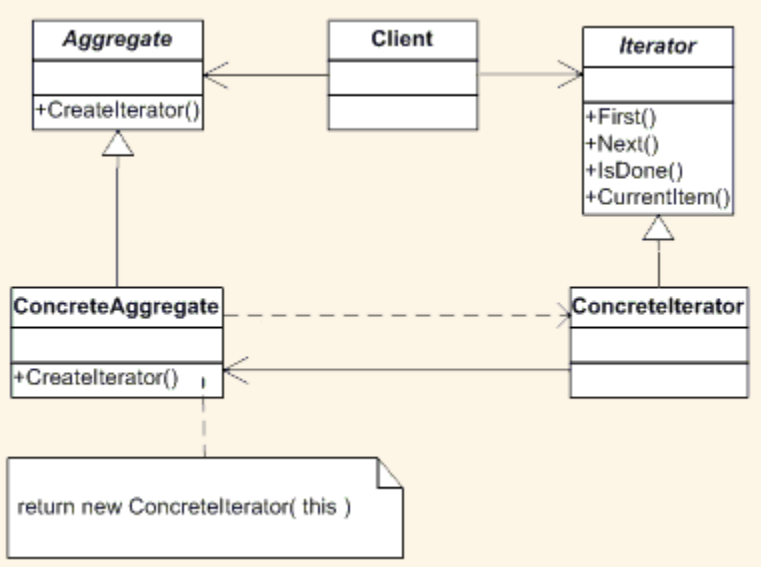
(NET) NET (2016)

Iterator in VB.NET

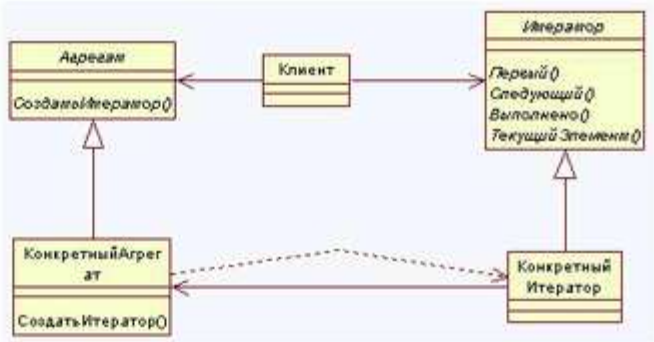
Sequentially access the elements of a collection

назад

Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation. Iterator pattern provides for a way to traverse (iterate) over a collection of items without detailing the underlying structure of the collection.



3.2.2 Итератор (Iterator) или Курсор (Cursor) - GoF

Проблема	Составной объект, например, список, должен предоставлять доступ к своим элементам (объектам), не раскрывая их внутреннюю структуру, причем перебирать список требуется по-разному в зависимости от задачи.
Решение	Создается класс "Итератор", коорый определяет интерфейс для доступа и перебора элементов, "КонкретныйИтератор" реализует интерфейсласса "Итератор" и следит за текущей позицией при обходе "Агрегата". "Агрегат" определяет интерфейс для создания объекта - итератора. "КонкретныйАгрегат" реализует интерфейс создания итератора и возвращает экземпляр класса "КонкретныйИтератор", "КонкретныйИтератор" отслеживает текущий объект в агрегате и может вычислить следующий объект при переборе. 
Преимущества	Поддерживает различные способы перебора агрегата, одновременно могут быть активны несколько переборов.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Iterator Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Iterator.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              Dim a As New ConcreteAggregate()
7:              a(0) = "Item A"
8:              a(1) = "Item B"
9:              a(2) = "Item C"
10:             a(3) = "Item D"
11:             ' Create Iterator and provide aggregate
12:             Dim i As Iterator = a.CreateIterator()
13:             Console.WriteLine("Iterating over collection:")
14:             Dim item As Object = i.First()
15:             While item IsNot Nothing
16:                 Console.WriteLine(item)
17:                 item = i.[Next]()
18:             End While
19:             ' Wait for user
20:             Console.ReadKey()
21:         End Sub
22:     End Class
23:
24:     ' The 'Aggregate' abstract class
25:     MustInherit Class Aggregate
26:         Public MustOverride Function CreateIterator() As Iterator
27:     End Class
28:
29:     ' The 'ConcreteAggregate' class
30:     Class ConcreteAggregate
31:         Inherits Aggregate
32:         Private _items As New ArrayList()
33:         Public Overrides Function CreateIterator() As Iterator
34:             Return New ConcreteIterator(Me)
35:         End Function
36:         ' Gets item count
```

```
37:         Public ReadOnly Property Count() As Integer
38:             Get
39:                 Return _items.Count
40:             End Get
41:         End Property
42:         ' Indexer
43:         Default Public Property Item(index As Integer) As Object
44:             Get
45:                 Return _items(index)
46:             End Get
47:             Set(value As Object)
48:                 _items.Insert(index, value)
49:             End Set
50:         End Property
51:     End Class
52:
53:     ' The 'Iterator' abstract class
54:     MustInherit Class Iterator
55:         Public MustOverride Function First() As Object
56:         Public MustOverride Function [Next]() As Object
57:         Public MustOverride Function IsDone() As Boolean
58:         Public MustOverride Function CurrentItem() As Object
59:     End Class
60:
61:     ' The 'ConcreteIterator' class
62:     Class ConcreteIterator
63:         Inherits Iterator
64:         Private _aggregate As ConcreteAggregate
65:         Private _current As Integer = 0
66:         ' Constructor
67:         Public Sub New(aggregate As ConcreteAggregate)
68:             Me._aggregate = aggregate
69:         End Sub
70:         ' Gets first iteration item
71:         Public Overrides Function First() As Object
72:             Return _aggregate(0)
73:         End Function
74:         ' Gets next iteration item
75:         Public Overrides Function [Next]() As Object
76:             Dim ret As Object = Nothing
77:             If _current < _aggregate.Count - 1 Then
78:                 _current = _current - 1
79:                 ret = _aggregate(_current)
80:             End If
81:             Return ret
82:         End Function
83:         ' Gets current iteration item
84:         Public Overrides Function CurrentItem() As Object
85:             Return _aggregate(_current)
86:         End Function
87:         ' Gets whether iterations are complete
88:         Public Overrides Function IsDone() As Boolean
89:             Return _current >= _aggregate.Count
90:         End Function
91:     End Class
```

```
Iterating over collection:
Item A
Item B
Item C
Item D
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Iterator.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> <>

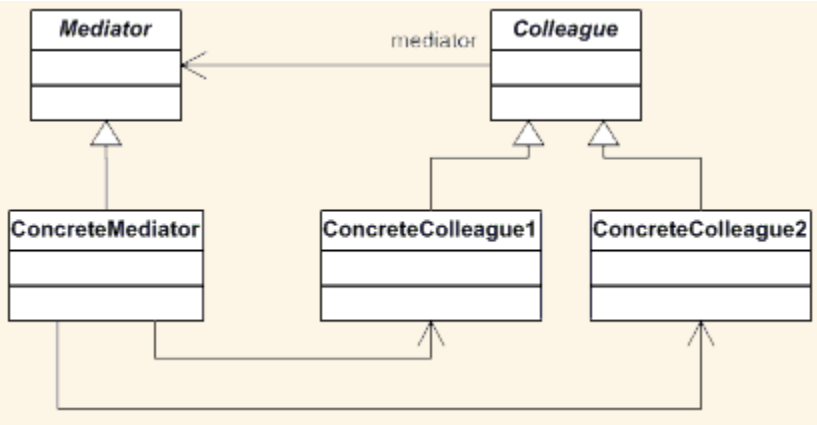
(NET) NET (2016)

Mediator in VB.NET

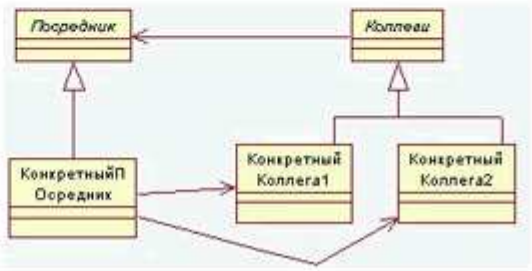
Defines simplified communication between classes

назад

Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently. Mediator pattern facilitate loosely coupled communication between different objects and object types. The mediator is a central hub through which all interaction must take place.



3.2.7 Посредник (Mediator) - GoF

Проблема	Обеспечить взаимодействие множества обьектов, сформировав при этом слабую связанность и избавив объекты от необходимости явно ссылаться друг на друга.
Решение	Создать объект инкапсулирующий способ взаимодействия множества обьектов.
Пример	"Посредник" определяет интерфейс для обмена информацией с объектами "Коллеги", "КонкретныйПосредник" координирует действия обьектов "Коллеги". Каждый класс "Коллеги" знает о своем объекте "Посредник", все "Коллеги" обмениваются информацией только с посредником, при его отсутствии им пришлось бы обмениваться информацией напрямую. "Коллеги" посылают запросы посреднику и получают запросы от него. "Посредник" реализует кооперативное поведения, пересылая каждый запрос одному или нескольким "Коллегам". 
Преимущества	Устраняется связанность между "Коллегами", централизуется управление.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Mediator Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Mediator.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              Dim m As New ConcreteMediator()
7:              Dim c1 As New ConcreteColleague1(m)
8:              Dim c2 As New ConcreteColleague2(m)
9:              m.Colleague1 = c1
10:             m.Colleague2 = c2
11:             c1.Send("How are you?")
12:             c2.Send("Fine, thanks")
13:             ' Wait for user
14:             Console.ReadKey()
15:         End Sub
16:     End Class
17:
18:     ' The 'Mediator' abstract class
19:     MustInherit Class Mediator
20:         Public MustOverride Sub Send(message As String, colleague As Colleague)
21:     End Class
22:
23:     ' The 'ConcreteMediator' class
24:     Class ConcreteMediator
25:         Inherits Mediator
26:         Private _colleague1 As ConcreteColleague1
27:         Private _colleague2 As ConcreteColleague2
28:         Public WriteOnly Property Colleague1() As ConcreteColleague1
29:             Set(value As ConcreteColleague1)
30:                 _colleague1 = value
31:             End Set
32:         End Property
33:         Public WriteOnly Property Colleague2() As ConcreteColleague2
34:             Set(value As ConcreteColleague2)
35:                 _colleague2 = value
36:             End Set
37:         End Property
38:         Public Overrides Sub Send(message As String, colleague As Colleague)
39:             If colleague Is _colleague1 Then
40:                 _colleague2.Notify(message)
41:             Else
42:                 _colleague1.Notify(message)
43:             End If
44:         End Sub
45:     End Class
46:
47:     ' The 'Colleague' abstract class
48:     MustInherit Class Colleague
49:         Protected mediator As Mediator
50:         ' Constructor
51:         Public Sub New(mediator As Mediator)
52:             Me.mediator = mediator
53:         End Sub
54:     End Class
55:
56:     ' A 'ConcreteColleague' class
57:     Class ConcreteColleague1
58:         Inherits Colleague
59:         ' Constructor
60:         Public Sub New(mediator As Mediator)
61:             MyBase.New(mediator)
62:         End Sub
63:         Public Sub Send(message As String)
64:             mediator.Send(message, Me)
65:         End Sub
66:         Public Sub Notify(message As String)
67:             Console.WriteLine(Convert.ToString("Colleague1 gets message: ") & message)
68:         End Sub
69:     End Class
70:
71:     ' A 'ConcreteColleague' class
72:     Class ConcreteColleague2
73:         Inherits Colleague
```

```
74:          ' Constructor
75:          Public Sub New(mediator As Mediator)
76:              MyBase.New(mediator)
77:          End Sub
78:          Public Sub Send(message As String)
79:              mediator.Send(message, Me)
80:          End Sub
81:          Public Sub Notify(message As String)
82:              Console.WriteLine(Convert.ToString("Colleague2 gets message: ") & message)
83:          End Sub
84:      End Class
```

```
Colleague2 gets message: How are you?
Colleague1 gets message: Fine, thanks
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Mediator.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

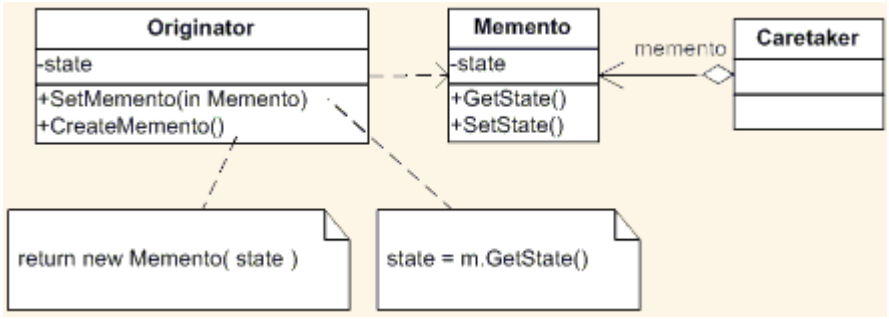
(NET) NET (2016)

Memento in VB.NET

Capture and restore an object's internal state

назад

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later. Memento pattern temporary saves and restores another object's internal state.



3.2.10 Хранитель (Memento) - GoF

Проблема	Необходимо зафиксировать поведение объекта для реализации, например, механизма отката.
Решение	Зафиксировать и вынести (не нарушая инкапсуляции) за пределы объекта его внутреннее состояние так, чтобы впоследствии можно было восстановить в нем объект. "Хранитель" сохраняет внутреннее состояние объекта "Хозяин" и запрещает доступ к себе всем другим объектам кроме "Хозяина", который имеет доступ ко всем данным для восстановления в прежнем состоянии. "Посыльный" может лишь передавать "Хранителя" другим объектам. "Хозяин" создает "Хранителя", содержащего снимок текущего внутреннего состояния и использует "Хранитель" для восстановления внутреннего состояния. "Посыльный" отвечает за сохранение "Хранителя", при этом не производит никаких операций над "Хранителем" и не исследует его внутреннее содержимое. "Посыльный" запрашивает "Хранитель" у "Хозяина", некоторое время держит его у себя, а затем возвращает "Хозяину". 
Преимущества	Не раскрывается информация, которая доступна только "Хозяину", упрощается структура "Хозяина".
Недостатки	С использованием "Хранителей" могут быть связаны значительные издержки, если "Хозяин" должен копировать большой объём информации, или если копирование должно проводиться часто.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Memento Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Memento.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              Dim o As New Originator()
7:              o.State = "On"
8:              ' Store internal state
9:              Dim c As New Caretaker()
10:             c.Memento = o.CreateMemento()
11:             ' Continue changing originator
12:             o.State = "Off"
13:             ' Restore saved state
14:             o.SetMemento(c.Memento)
15:             ' Wait for user
16:             Console.ReadKey()
17:         End Sub
18:     End Class
19:
20:     ' The 'Originator' class
21:     Class Originator
22:         Private _state As String
23:         ' Property
24:         Public Property State() As String
25:             Get
26:                 Return _state
27:             End Get
28:             Set(value As String)
29:                 _state = value
30:                 Console.WriteLine(Convert.ToString("State = ") & _state)
31:             End Set
32:         End Property
33:         ' Creates memento
34:         Public Function CreateMemento() As Memento
35:             Return (New Memento(_state))
36:         End Function
37:         ' Restores original state
38:         Public Sub SetMemento(memento As Memento)
39:             Console.WriteLine("Restoring state...")
40:             State = memento.State
41:         End Sub
42:     End Class
43:
44:     ' The 'Memento' class
45:     Class Memento
46:         Private _state As String
47:         ' Constructor
48:         Public Sub New(state As String)
49:             Me._state = state
50:         End Sub
51:         ' Gets or sets state
52:         Public ReadOnly Property State() As String
53:             Get
54:                 Return _state
55:             End Get
56:         End Property
57:     End Class
58:
59:     ' The 'Caretaker' class
60:     Class Caretaker
61:         Private _memento As Memento
62:         ' Gets or sets memento
63:         Public Property Memento() As Memento
64:             Get
65:                 Return _memento
66:             End Get
67:             Set(value As Memento)
68:                 _memento = value
69:             End Set
70:         End Property
71:     End Class
```

```
State = On
State = Off
Restoring state:
State = On
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Memento.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

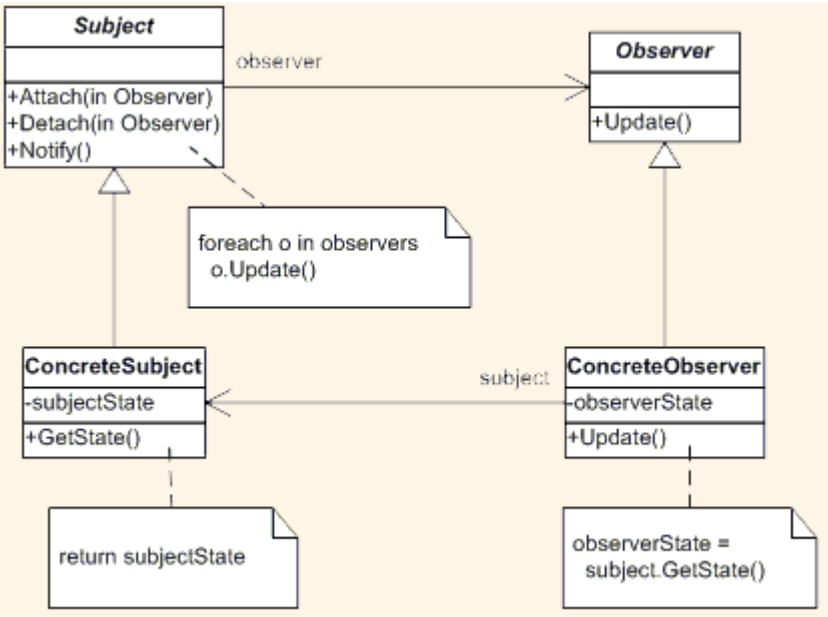
(NET) NET (2016)

Observer in VB.NET

A way of notifying change to a number of classes

назад

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. Observer pattern with registered objects are notified of and updated with a state change.



3.2.4 Наблюдатель (Observer), Опубликовать - подписаться (Publish - Subscribe) или Delegation Event Model - GoF

Проблема	Один объект ("Подписчик") должен знать об изменении состояний или некоторых событиях другого объекта. При этом необходимо поддерживать низкий уровень связывания с объектом - "Подписчиком".
Решение	Определить интерфейс "Подписки". Объекты - подписчики реализуют этот интерфейс и динамически регистрируются для получении информации о некотором событии. Затем при реализации условленного события оповещаются все объекты - подписчики.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Observer Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Observer.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              ' Configure Observer pattern
7:              Dim s As New ConcreteSubject()
8:              s.Attach(New ConcreteObserver(s, "X"))
9:              s.Attach(New ConcreteObserver(s, "Y"))
```

```
10:             s.Attach(New ConcreteObserver(s, "Z"))
11:             ' Change subject and notify observers
12:             s.SubjectState = "ABC"
13:             s.Notify()
14:             ' Wait for user
15:             Console.ReadKey()
16:         End Sub
17:     End Class
18:
19:     ' The 'Subject' abstract class
20:     MustInherit Class Subject
21:         Private _observers As New List(Of Observer)()
22:         Public Sub Attach(observer As Observer)
23:             _observers.Add(observer)
24:         End Sub
25:         Public Sub Detach(observer As Observer)
26:             _observers.Remove(observer)
27:         End Sub
28:         Public Sub Notify()
29:             For Each o As Observer In _observers
30:                 o.Update()
31:             Next
32:         End Sub
33:     End Class
34:
35:     ' The 'ConcreteSubject' class
36:     Class ConcreteSubject
37:         Inherits Subject
38:         Private _subjectState As String
39:         ' Gets or sets subject state
40:         Public Property SubjectState() As String
41:             Get
42:                 Return _subjectState
43:             End Get
44:             Set(value As String)
45:                 _subjectState = value
46:             End Set
47:         End Property
48:     End Class
49:
50:     ' The 'Observer' abstract class
51:     MustInherit Class Observer
52:         Public MustOverride Sub Update()
53:     End Class
54:
55:     ' The 'ConcreteObserver' class
56:     Class ConcreteObserver
57:         Inherits Observer
58:         Private _name As String
59:         Private _observerState As String
60:         Private _subject As ConcreteSubject
61:         ' Constructor
62:         Public Sub New(subject As ConcreteSubject, name As String)
63:             Me._subject = subject
64:             Me._name = name
65:         End Sub
66:         Public Overrides Sub Update()
67:             _observerState = _subject.SubjectState
68:             Console.WriteLine("Observer {0}'s new state is {1}", _name, _observerState)
69:         End Sub
70:         ' Gets or sets subject
71:         Public Property Subject() As ConcreteSubject
72:             Get
73:                 Return _subject
74:             End Get
75:             Set(value As ConcreteSubject)
76:                 _subject = value
77:             End Set
78:         End Property
79:     End Class
```

```
Observer X's new state is ABC
Observer Y's new state is ABC
Observer Z's new state is ABC
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Observer.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

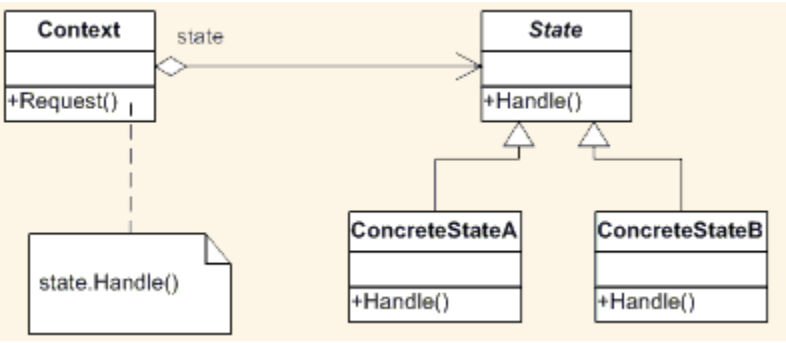
(NET) NET (2016)

State in VB.NET

Alter an object's behavior when its state changes

назад

Allow an object to alter its behavior when its internal state changes. The object will appear to change its class. State pattern allows an object to behave differently depending on its internal state. The difference in behavior is delegated to objects that represent this state.



3.2.8 Состояние (State) - GoF

Проблема	Варьировать поведение объекта в зависимости от его внутреннего состояния
Решение	Класс "Контекст" делегирует зависящие от состояния запросы текущему объекту "КонкретноеСостояние" (хранит экземпляр подкласса "КонкретноеСостояние", которым определяется текущее состояние), и определяет интерфейс, представляющий интерес для клиентов. "КонкретноеСостояние" реализует поведение, ассоциированное с неким состоянием объекта "Контекст". "Состояние" определяет интерфейс для инкапсуляции поведения, ассоциированного с конкретным экземпляром "Контекста". 
Преимущества	Локализует зависящее от состояния поведение и делит его на части, соответствующие состояниям, переходы между состояниями становятся явными.

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' State Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/State.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
```

```
            ' Setup context in a state
            Dim c As New Context(New ConcreteStateA())
            ' Issue requests, which toggles state
            c.Request()
            c.Request()
            c.Request()
            c.Request()
            ' Wait for user
            Console.ReadKey()
        End Sub
    End Class

    ' The 'State' abstract class
    MustInherit Class State
        Public MustOverride Sub Handle(context As Context)
    End Class

    ' A 'ConcreteState' class
    Class ConcreteStateA
        Inherits State
        Public Overrides Sub Handle(context As Context)
            context.State = New ConcreteStateB()
        End Sub
    End Class

    ' A 'ConcreteState' class
    Class ConcreteStateB
        Inherits State
        Public Overrides Sub Handle(context As Context)
            context.State = New ConcreteStateA()
        End Sub
    End Class

    ' The 'Context' class
    Class Context
        Private _state As State
        ' Constructor
        Public Sub New(state As State)
            Me.State = state
        End Sub
        ' Gets or sets the state
        Public Property State() As State
            Get
                Return _state
            End Get
            Set(value As State)
                _state = value
                Console.WriteLine("State: " + _state.[GetType]().Name)
            End Set
        End Property
        Public Sub Request()
            _state.Handle(Me)
        End Sub
    End Class
```

```
State: ConcreteStateA
State: ConcreteStateB
State: ConcreteStateA
State: ConcreteStateB
State: ConcreteStateA
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation

- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/State.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

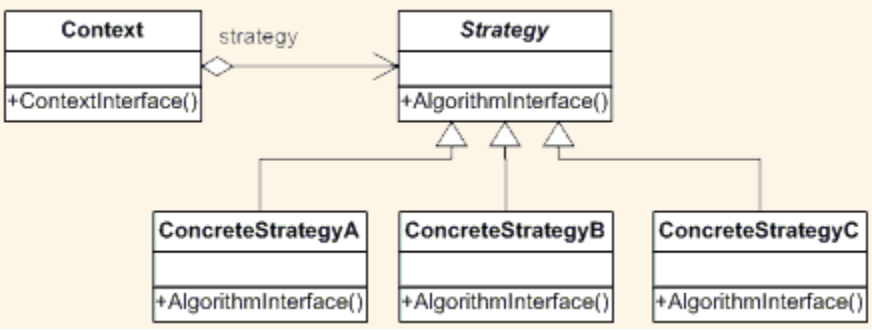
(NET) NET (2016)

Strategy in VB.NET

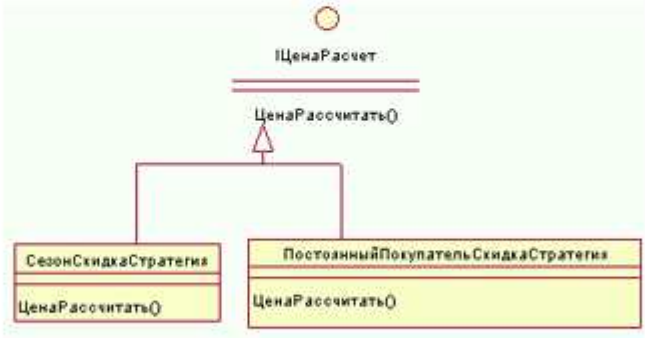
Encapsulates an algorithm inside a class

назад

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. Strategy pattern encapsulate functionality in the form of an object. This allows clients to dynamically change algorithmic strategies.



3.2.9 Стратегия (Strategy) - GoF

Проблема	Спроектировать изменяемые, но надежные алгоритмы или стратегии.
Решение	Определить для каждого алгоритма или стратегии отдельный класс со стандартным интерфейсом.
Пример	Обеспечение сложной логики вычисления стоимости товаров с учетом сезонных скидок, скидок постоянным клиентам и т. п. Данная стратегия может изменяться.  Создается несколько классов "Стратегия", каждый из которых содержит один и тот же полиморфный метод "ЦенаРассчитать". В качестве параметров в этот метод передаются данные о продаже. Объект стратегии связывается с контекстным объектом (тем объектом, к которому применяется алгоритм).

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Strategy Design Pattern.
```

```
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Strategy.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              Dim context As Context
7:              ' Three contexts following different strategies
8:              context = New Context(New ConcreteStrategyA())
9:              context.ContextInterface()
10:             context = New Context(New ConcreteStrategyB())
11:             context.ContextInterface()
12:             context = New Context(New ConcreteStrategyC())
13:             context.ContextInterface()
14:             ' Wait for user
15:             Console.ReadKey()
16:         End Sub
17:     End Class
18:
19:     ' The 'Strategy' abstract class
20:     MustInherit Class Strategy
21:         Public MustOverride Sub AlgorithmInterface()
22:     End Class
23:
24:     ' A 'ConcreteStrategy' class
25:     Class ConcreteStrategyA
26:         Inherits Strategy
27:         Public Overrides Sub AlgorithmInterface()
28:             Console.WriteLine("Called ConcreteStrategyA.AlgorithmInterface()")
29:         End Sub
30:     End Class
31:
32:     ' A 'ConcreteStrategy' class
33:     Class ConcreteStrategyB
34:         Inherits Strategy
35:         Public Overrides Sub AlgorithmInterface()
36:             Console.WriteLine("Called ConcreteStrategyB.AlgorithmInterface()")
37:         End Sub
38:     End Class
39:
40:     ' A 'ConcreteStrategy' class
41:     Class ConcreteStrategyC
42:         Inherits Strategy
43:         Public Overrides Sub AlgorithmInterface()
44:             Console.WriteLine("Called ConcreteStrategyC.AlgorithmInterface()")
45:         End Sub
46:     End Class
47:
48:     ' The 'Context' class
49:     Class Context
50:         Private _strategy As Strategy
51:         ' Constructor
52:         Public Sub New(strategy As Strategy)
53:             Me._strategy = strategy
54:         End Sub
55:         Public Sub ContextInterface()
56:             _strategy.AlgorithmInterface()
57:         End Sub
58:     End Class
```

```
Called ConcreteStrategyA.AlgorithmInterface()
Called ConcreteStrategyB.AlgorithmInterface()
Called ConcreteStrategyC.AlgorithmInterface()
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation

- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Strategy.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

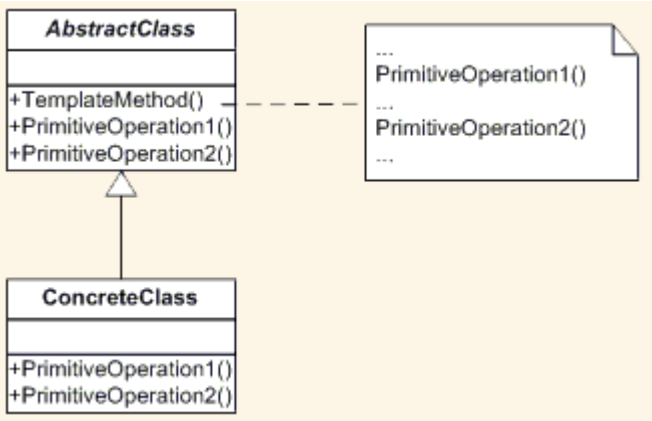
(NET) NET (2016)

Template Method in VB.NET

Defer the exact steps of an algorithm to a subclass

назад

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. Template method provide a skeleton calling sequence of methods. One or more steps can be deferred to subclasses which implement these steps without changing the overall calling sequence.



3.2.12 Шаблонный метод (Template Method) - GoF

Проблема	Определить алгоритм и реализовать возможность переопределения некоторых шагов алгоритма для подклассов (без изменения общей структуры алгоритма.
Решение	"АбстрактныйКласс" определяет абстрактные Операции(), замещаемые в конкретных подклассах для реализации шагов алгоритма, и реализует ШаблонныйМетод(), определяющий "скелет" алгоритма. "КонкретныйКласс" релизует Операции(), выполняющие шаги алгоритма способом, который зависит от подкласса. "КонкретныйКласс" предполагает, что инвариантные шаги алгоритма будут выполнены в "АбстрактномКлассе". 

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Template Method Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/TemplateMethod.htm
3:      Class MainApp
4:          ' Entry point into console application.
5:          Public Shared Sub Main()
6:              Dim aA As AbstractClass = New ConcreteClassA()
7:              aA.TemplateMethod()
8:              Dim aB As AbstractClass = New ConcreteClassB()
9:              aB.TemplateMethod()
10:             ' Wait for user
11:             Console.ReadKey()
12:         End Sub
13:     End Class
14:
15:     ' The 'AbstractClass' abstract class
16:     MustInherit Class AbstractClass
17:         Public MustOverride Sub PrimitiveOperation1()
18:         Public MustOverride Sub PrimitiveOperation2()
19:         ' The "Template method"
20:         Public Sub TemplateMethod()
21:             PrimitiveOperation1()
22:             PrimitiveOperation2()
23:             Console.WriteLine("")
24:         End Sub
25:     End Class
26:
27:     ' A 'ConcreteClass' class
28:     Class ConcreteClassA
29:         Inherits AbstractClass
30:         Public Overrides Sub PrimitiveOperation1()
31:             Console.WriteLine("ConcreteClassA.PrimitiveOperation1()")
32:         End Sub
33:         Public Overrides Sub PrimitiveOperation2()
34:             Console.WriteLine("ConcreteClassA.PrimitiveOperation2()")
35:         End Sub
36:     End Class
37:
38:     ' A 'ConcreteClass' class
39:     Class ConcreteClassB
40:         Inherits AbstractClass
41:         Public Overrides Sub PrimitiveOperation1()
42:             Console.WriteLine("ConcreteClassB.PrimitiveOperation1()")
43:         End Sub
44:         Public Overrides Sub PrimitiveOperation2()
45:             Console.WriteLine("ConcreteClassB.PrimitiveOperation2()")
46:         End Sub
47:     End Class
```

```
ConcreteClassA.PrimitiveOperation1()
ConcreteClassA.PrimitiveOperation2()

ConcreteClassB.PrimitiveOperation1()
ConcreteClassB.PrimitiveOperation2()
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program

- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/TemplateMethod.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>

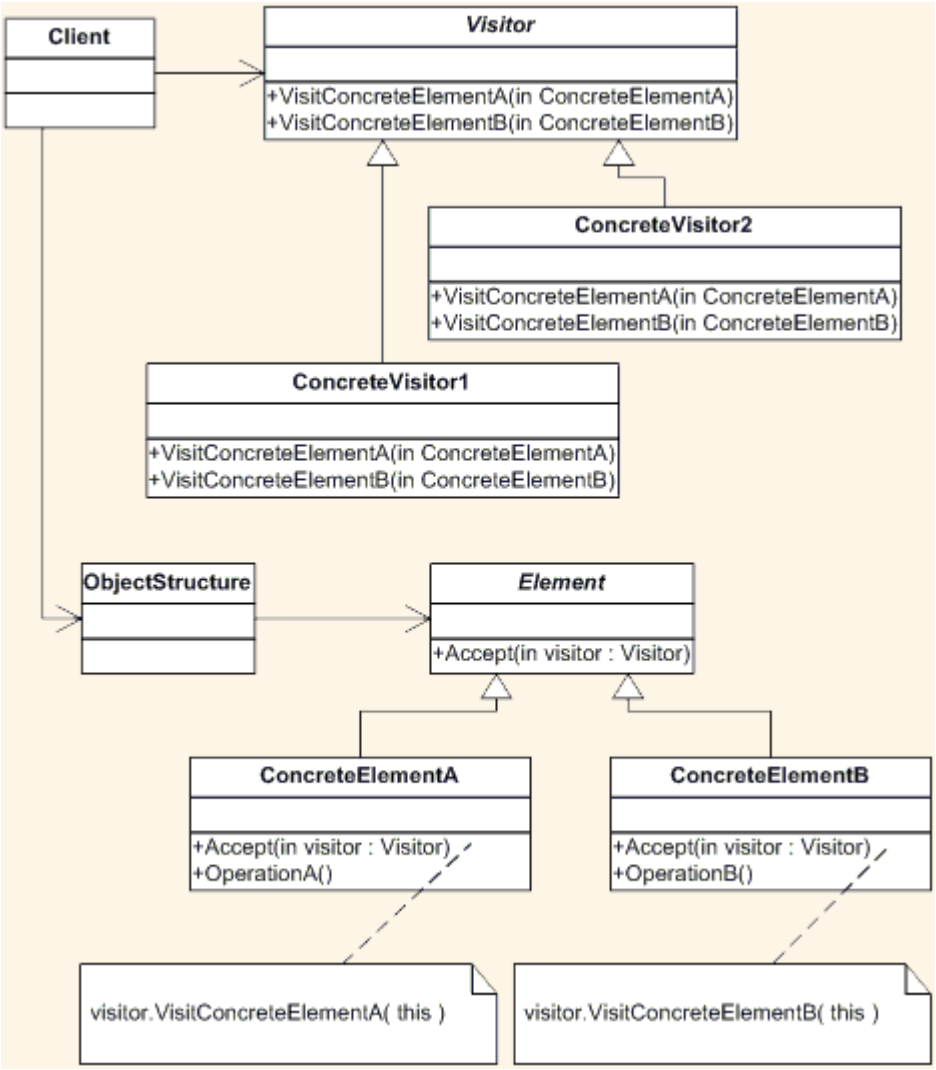
(NET) NET (2016)

Visitor in VB.NET

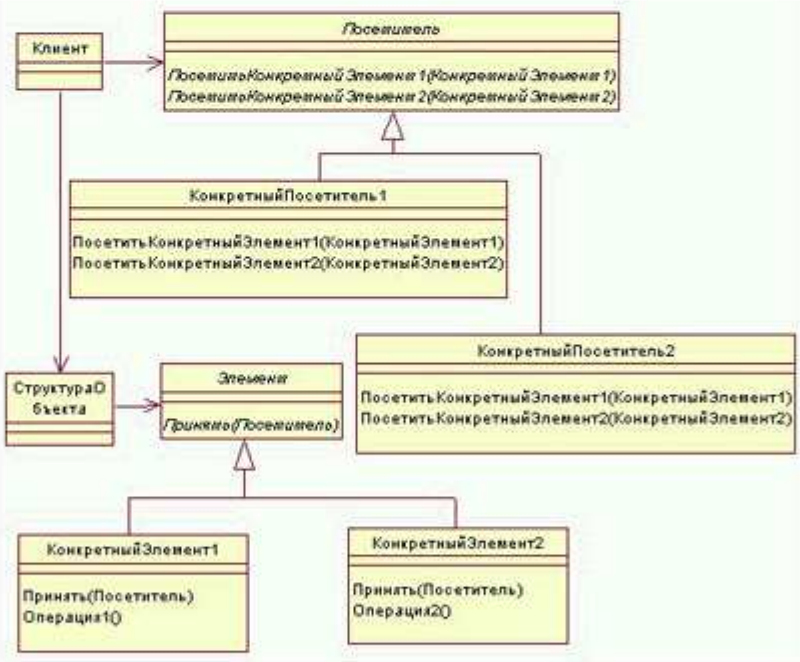
Defines a new operation to a class without change

назад

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates. Different visitor objects define different operations.



3.2.6 Посетитель (Visitor) - GoF

Проблема	Над каждым объектом некоторой структуры выполняется операция. Определить новую операцию, не изменяя классы объектов.
Решение	Клиент, использующий данный паттерн, должен создать объект класса "КонкретныйПосетитель", а затем посетить каждый элемент структуры. "Посетитель" объявляет операцию "Посетить" для каждого класса "КонкретныйЭлемент" (имя и сигнатура данной операции идентифицируют класс, элемент которого посещает "Посетитель" - то есть, посетитель может обращаться к элементу напрямую). "КонкретныйПосетитель" реализует все операции, обьявленные в классе "Посетитель". Каждая операция реализует фрагмент алгоритма, определенного для класса соответствующего объекта в структуре. Класс "КонкретныйПосетитель"предоставляет контекст для этого алгоритма и сохраняет его локальное состояние. "Элемент" определяет операцию "Принять", которая принимает "Посетителя" в качестве аргумента, "КонкретныйЭлемент" реализует операцию "Принять", которая принимает "Посетителя" в качестве аргумента. "СтруктураОбьекта" может перечислить свои аргументы и предоставить посетителю высокоуровневый интерфейс для посещения своих элементов. 
Рекомендации	Логично использовать, если в структуре присутствуют объекты многих классов с различными интерфейсами, и необходимо выполнить над ними операции, зависящие от конкретных классов, или если классы, устанавливающие структуру обьектов изменяются редко, но новые операции над этой структурой добавляются часто.
Преимущества	Упрощается добавление новых операций, объединяет родственные операции в классе "Посетитель".
Недостатки	Затруднено добавление новых классов "КонкретныйЭлемент", поскольку требуется объявление новой абстрактной операции в классе "Посетитель".

Please download project with this source code from https://github.com/ViacheslavUKR/StandardDisignOopPattern

```
1:      ' Visitor Design Pattern.
2:      ' See description in http://www.vb-net.ru/ProgramTheory/Visitor.htm
3:      Class MainApp
4:          Public Shared Sub Main()
5:              ' Setup structure
```

```
 6:              Dim o As New ObjectStructure()
 7:              o.Attach(New ConcreteElementA())
 8:              o.Attach(New ConcreteElementB())
 9:              ' Create visitor objects
10:              Dim v1 As New ConcreteVisitor1()
11:              Dim v2 As New ConcreteVisitor2()
12:              ' Structure accepting visitors
13:              o.Accept(v1)
14:              o.Accept(v2)
15:              ' Wait for user
16:              Console.ReadKey()
17:          End Sub
18:      End Class
19:
20:      ' The 'Visitor' abstract class
21:      MustInherit Class Visitor
22:          Public MustOverride Sub VisitConcreteElementA(concreteElementA As ConcreteElementA)
23:          Public MustOverride Sub VisitConcreteElementB(concreteElementB As ConcreteElementB)
24:      End Class
25:
26:      ' A 'ConcreteVisitor' class
27:      Class ConcreteVisitor1
28:          Inherits Visitor
29:          Public Overrides Sub VisitConcreteElementA(concreteElementA As ConcreteElementA)
30:              Console.WriteLine("{0} visited by {1}", concreteElementA.[GetType]().Name, Me.[GetType]().Name)
31:          End Sub
32:          Public Overrides Sub VisitConcreteElementB(concreteElementB As ConcreteElementB)
33:              Console.WriteLine("{0} visited by {1}", concreteElementB.[GetType]().Name, Me.[GetType]().Name)
34:          End Sub
35:      End Class
36:
37:      ' A 'ConcreteVisitor' class
38:      Class ConcreteVisitor2
39:          Inherits Visitor
40:          Public Overrides Sub VisitConcreteElementA(concreteElementA As ConcreteElementA)
41:              Console.WriteLine("{0} visited by {1}", concreteElementA.[GetType]().Name, Me.[GetType]().Name)
42:          End Sub
43:          Public Overrides Sub VisitConcreteElementB(concreteElementB As ConcreteElementB)
44:              Console.WriteLine("{0} visited by {1}", concreteElementB.[GetType]().Name, Me.[GetType]().Name)
45:          End Sub
46:      End Class
47:
48:      ' The 'Element' abstract class
49:      MustInherit Class Element
50:          Public MustOverride Sub Accept(visitor As Visitor)
51:      End Class
52:
53:      ' A 'ConcreteElement' class
54:      Class ConcreteElementA
55:          Inherits Element
56:          Public Overrides Sub Accept(visitor As Visitor)
57:              visitor.VisitConcreteElementA(Me)
58:          End Sub
59:          Public Sub OperationA()
60:          End Sub
61:      End Class
62:
63:      ' A 'ConcreteElement' class
64:      Class ConcreteElementB
65:          Inherits Element
66:          Public Overrides Sub Accept(visitor As Visitor)
67:              visitor.VisitConcreteElementB(Me)
68:          End Sub
69:          Public Sub OperationB()
70:          End Sub
71:      End Class
72:
73:      ' The 'ObjectStructure' class
74:      Class ObjectStructure
75:          Private _elements As New List(Of Element)()
76:          Public Sub Attach(element As Element)
77:              _elements.Add(element)
78:          End Sub
79:          Public Sub Detach(element As Element)
80:              _elements.Remove(element)
```

```
81:          End Sub
82:          Public Sub Accept(visitor As Visitor)
83:              For Each element As Element In _elements
84:                  element.Accept(visitor)
85:              Next
86:          End Sub
87:      End Class
```

```
ConcreteElementA visited by ConcreteVisitor1
ConcreteElementB visited by ConcreteVisitor1
ConcreteElementA visited by ConcreteVisitor2
ConcreteElementB visited by ConcreteVisitor2
```

See also:

Creational Patterns

- Abstract Factory in VB.NET - Creates an instance of several families of classes.
- Builder in VB.NET - Separates object construction from its representation.
- Factory Method in VB.NET - Creates an instance of several derived classes.
- Prototype in VB.NET - A fully initialized instance to be copied or cloned.
- Singleton in VB.NET - A class of which only a single instance can exist.

Structural Patterns

- Adapter in VB.NET - Match interfaces of different classes
- Bridge in VB.NET - Separates an object's interface from its implementation
- Composite in VB.NET - A tree structure of simple and composite objects
- Decorator in VB.NET - Add responsibilities to objects dynamically
- Facade in VB.NET - A single class that represents an entire subsystem
- Flyweight in VB.NET - A fine-grained instance used for efficient sharing
- Proxy in VB.NET - An object representing another object

Behavioral Patterns

- Chain of Resp. in VB.NET - A way of passing a request between a chain of objects
- Command in VB.NET - Encapsulate a command request as an object
- Interpreter in VB.NET - A way to include language elements in a program
- Iterator in VB.NET - Sequentially access the elements of a collection
- Mediator in VB.NET - Defines simplified communication between classes
- Memento in VB.NET - Capture and restore an object's internal state
- Observer in VB.NET - A way of notifying change to a number of classes
- State in VB.NET - Alter an object's behavior when its state changes
- Strategy in VB.NET - Encapsulates an algorithm inside a class
- Template Method in VB.NET - Defer the exact steps of an algorithm to a subclass
- Visitor in VB.NET - Defines a new operation to a class without change

Комментарии к этой страничке (0) ссылка на эту страничку: http://www.vb-net.ru/ProgramTheory/Visitor.htm

<На главную> <В раздел ASP> <В раздел NET> <В раздел SQL> <В раздел Разное> <Написать автору> < Поблагодарить>