Login (/login.aspx) Join Now (/join.aspx)

(/)

back to .NET Design Patterns (/net/design-patterns)
(/net/design-patterns)

Memento

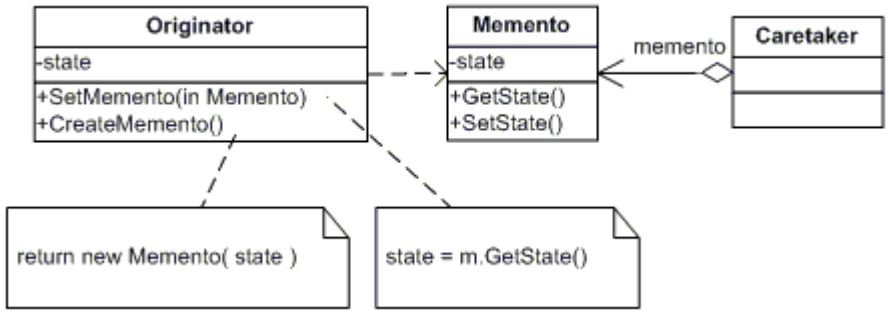
Definition

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

Frequency of use: 

Low

UML class diagram



Participants

The classes and objects participating in this pattern are:

- **Memento (Memento)**
  - stores internal state of the Originator object. The memento may store as much or as little of the originator's internal state as necessary at its originator's discretion.
  - protect against access by objects of other than the originator. Mementos have effectively two interfaces. Caretaker sees a narrow interface to the Memento -- it can only pass the memento to the other objects. Originator, in contrast, sees a wide interface, one that lets it access all the data necessary to restore itself to its previous state. Ideally, only the originator that produces the memento would be permitted to access the memento's internal state.
- **Originator (SalesProspect)**
  - creates a memento containing a snapshot of its current internal state.
  - uses the memento to restore its internal state
- **Caretaker (Caretaker)**
  - is responsible for the memento's safekeeping
  - never operates on or examines the contents of a memento.

Structural code in C#

This structural code demonstrates the Memento pattern which temporary saves and restores another object's internal state.

```
1.
2.
3.  using System;
4.
5.  namespace DoFactory.GangOfFour.Memento.Structural
6.  {
7.    /// <summary>
8.    /// MainApp startup class for Structural
9.    /// Memento Design Pattern.
10.   /// </summary>
11.   class MainApp
12.   {
13.     /// <summary>
14.     /// Entry point into console application.
15.     /// </summary>
16.     static void Main()
17.     {
18.       Originator o = new Originator();
19.       o.State = "On";
20.
21.       // Store internal state
22.       Caretaker c = new Caretaker();
23.       c.Memento = o.CreateMemento();
24.
25.       // Continue changing originator
26.       o.State = "Off";
27.
28.       // Restore saved state
29.       o.SetMemento(c.Memento);
30.
31.       // Wait for user
32.       Console.ReadKey();
33.     }
34.   }
35.
36.   /// <summary>
37.   /// The 'Originator' class
38.   /// </summary>
39.   class Originator
40.   {
```

```
41.       private string _state;
42.
43.       // Property
44.       public string State
45.       {
46.         get { return _state; }
47.         set
48.         {
49.           _state = value;
50.           Console.WriteLine("State = " + _state);
51.         }
52.       }
53.
54.       // Creates memento
55.       public Memento CreateMemento()
56.       {
57.         return (new Memento(_state));
58.       }
59.
60.       // Restores original state
61.       public void SetMemento(Memento memento)
62.       {
63.         Console.WriteLine("Restoring state...");
64.         State = memento.State;
65.       }
66.     }
67.
68.     /// <summary>
69.     /// The 'Memento' class
70.     /// </summary>
71.     class Memento
72.     {
73.       private string _state;
74.
75.       // Constructor
76.       public Memento(string state)
77.       {
78.         this._state = state;
79.       }
80.
81.       // Gets or sets state
```

```
82.       public string State
83.       {
84.         get { return _state; }
85.       }
86.     }
87.
88.     /// <summary>
89.     /// The 'Caretaker' class
90.     /// </summary>
91.     class Caretaker
92.     {
93.       private Memento _memento;
94.
95.       // Gets or sets memento
96.       public Memento Memento
97.       {
98.         set { _memento = value; }
99.         get { return _memento; }
100.      }
101.    }
102.  }
103.
104.
105.
```

Output

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

Real-world code in C#

This real-world code demonstrates the Memento pattern which temporarily saves and then restores the SalesProspect's internal state.

```
1.
2.
3.  using System;
4.
5.  namespace DoFactory.GangOfFour.Memento.RealWorld
6.  {
7.    /// <summary>
8.    /// MainApp startup class for Real-World
9.    /// Memento Design Pattern.
10.   /// </summary>
11.   class MainApp
12.   {
13.     /// <summary>
14.     /// Entry point into console application.
15.     /// </summary>
16.     static void Main()
17.     {
18.       SalesProspect s = new SalesProspect();
19.       s.Name = "Noel van Halen";
20.       s.Phone = "(412) 256-0990";
21.       s.Budget = 25000.0;
22.
23.       // Store internal state
24.       ProspectMemory m = new ProspectMemory();
25.       m.Memento = s.SaveMemento();
26.
27.       // Continue changing originator
28.       s.Name = "Leo Welch";
29.       s.Phone = "(310) 209-7111";
30.       s.Budget = 1000000.0;
31.
32.       // Restore saved state
33.       s.RestoreMemento(m.Memento);
34.
35.       // Wait for user
36.       Console.ReadKey();
37.     }
38.   }
39.
40.   /// <summary>
```

```
41.     /// The 'Originator' class
42.     /// </summary>
43.     class SalesProspect
44.     {
45.       private string _name;
46.       private string _phone;
47.       private double _budget;
48.
49.       // Gets or sets name
50.       public string Name
51.       {
52.         get { return _name; }
53.         set
54.         {
55.           _name = value;
56.           Console.WriteLine("Name:  " + _name);
57.         }
58.       }
59.
60.       // Gets or sets phone
61.       public string Phone
62.       {
63.         get { return _phone; }
64.         set
65.         {
66.           _phone = value;
67.           Console.WriteLine("Phone: " + _phone);
68.         }
69.       }
70.
71.       // Gets or sets budget
72.       public double Budget
73.       {
74.         get { return _budget; }
75.         set
76.         {
77.           _budget = value;
78.           Console.WriteLine("Budget: " + _budget);
79.         }
80.       }
81.
```

```
82.       // Stores memento
83.       public Memento SaveMemento()
84.       {
85.         Console.WriteLine("\nSaving state --\n");
86.         return new Memento(_name, _phone, _budget);
87.       }
88.
89.       // Restores memento
90.       public void RestoreMemento(Memento memento)
91.       {
92.         Console.WriteLine("\nRestoring state --\n");
93.         this.Name = memento.Name;
94.         this.Phone = memento.Phone;
95.         this.Budget = memento.Budget;
96.       }
97.     }
98.
99.     /// <summary>
100.    /// The 'Memento' class
101.    /// </summary>
102.    class Memento
103.    {
104.      private string _name;
105.      private string _phone;
106.      private double _budget;
107.
108.      // Constructor
109.      public Memento(string name, string phone, double budget)
110.      {
111.        this._name = name;
112.        this._phone = phone;
113.        this._budget = budget;
114.      }
115.
116.      // Gets or sets name
117.      public string Name
118.      {
119.        get { return _name; }
120.        set { _name = value; }
121.      }
122.
```

```
123.       // Gets or set phone
124.       public string Phone
125.       {
126.         get { return _phone; }
127.         set { _phone = value; }
128.       }
129.
130.       // Gets or sets budget
131.       public double Budget
132.       {
133.         get { return _budget; }
134.         set { _budget = value; }
135.       }
136.     }
137.
138.     /// <summary>
139.     /// The 'Caretaker' class
140.     /// </summary>
141.     class ProspectMemory
142.     {
143.       private Memento _memento;
144.
145.       // Property
146.       public Memento Memento
147.       {
148.         set { _memento = value; }
149.         get { return _memento; }
150.       }
151.     }
152.   }
153.
154.
155.
```

Output

```
Name:   Noel van Halen
Phone:  (412) 256-0990
Budget: 25000
```

```
Saving state --

Name:   Leo Welch
Phone:  (310) 209-7111
Budget: 1000000

Restoring state --

Name:   Noel van Halen
Phone:  (412) 256-0990
Budget: 25000
```

.NET Optimized code in C#

The .NET optimized code demonstrates the same real-world situation as above but uses modern, built-in .NET features, such as, generics, reflection, object initializers, automatic properties, etc. You can find an example on our Singleton (/net/singleton-design-pattern#net) pattern page.

All other patterns (and much more) are available in our **.NET Design Pattern Framework 4.5.**

Not only does the **.NET Design Pattern Framework 4.5** cover GOF and Enterprise patterns, it also includes .NET pattern architectures that reduce the code you need to write by up to 75%. This unique package will change your .NET lifestyle -- for only $79. Here's what is included:

(/products/net-design-pattern-framework)

Two editions: C# and VB.NET (/products/net-design-pattern-framework)

Learn More (/products/net-design-pattern-framework)

- 69 gang-of-four pattern projects
- 46 head-first pattern projects
- Fowler's enterprise patterns
- Multi-tier patterns
- Convention over configuration
- Active Record and CQRS patterns
- Repository and Unit-of-Work patterns
- MVC, MVP, & MVVM patterns
- REST patterns with Web API
- Spark™ Rapid App Dev (RAD) platform!
- Art Shop MVC Reference Application
- 100% pure source code

Company

- About Us (/about)
- Our Story (/story)
- Services (/services)
- Training (/training)
- Contact Us (/contact)

- Privacy (/privacy)
- End User License (/eula)
- Terms (/terms)
- Licensing (/licensing)

Customers

- Our Customers (/customers)
- Customer Stories (/customers/stories)

Community

- Questions (/topic/search.aspx)
- Explore (/topic/topics.aspx)
- Tags (/tag/tags.aspx)

Reference Guides

- .NET Design Patterns (/net/design-patterns)
- JavaScript Design Patterns (/javascript/design-patterns)

- JavaScript Tutorial (/tutorial/javascript)
- SQL Tutorial (/sql/tutorial)

- Connection Strings (/reference/connection-strings)
- Visual Studio Shortcuts (/reference/visual-studio-shortcuts)
- C# Coding Standards (/reference/csharp-coding-standards)
- HTML Colors (/reference/html-color-codes)

Our Products

- .NET Design Pattern Framework (/products/net-design-pattern-framework) TM

- PRO .NET Design Pattern Framework (/products/pro-net-design-pattern-framework) TM

- JavaScript + jQuery Pattern Framework (/products/javascript-jquery-design-pattern-framework) TM
- SQL + Database Pattern Framework (/products/sql-database-design-pattern-framework) TM

- Products and Pricing (/products)

© 2019 - Data & Object Factory, LLC. dofactory.com. All rights reserved.