(/)

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

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

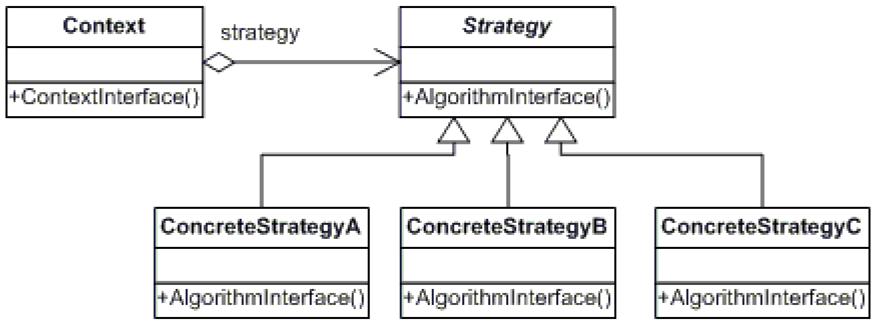
Strategy

Definition

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Frequency of use: 🟥🟥🟥🟥🟨 Medium high

UML class diagram



Participants

The classes and objects participating in this pattern are:

- **Strategy (SortStrategy)**
  - declares an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a ConcreteStrategy
- **ConcreteStrategy (QuickSort, ShellSort, MergeSort)**
  - implements the algorithm using the Strategy interface
- **Context (SortedList)**
  - is configured with a ConcreteStrategy object
  - maintains a reference to a Strategy object
  - may define an interface that lets Strategy access its data.

Structural code in C#

This structural code demonstrates the Strategy pattern which encapsulates functionality in the form of an object. This allows clients to dynamically change algorithmic strategies.

```


using System;

namespace DoFactory.GangOfFour.Strategy.Structural
{
  /// <summary>
  /// MainApp startup class for Structural
  /// Strategy Design Pattern.
  /// </summary>
  class MainApp
  {
    /// <summary>
    /// Entry point into console application.
    /// </summary>
    static void Main()
    {
      Context context;

      // Three contexts following different strategies
      context = new Context(new ConcreteStrategyA());
      context.ContextInterface();

      context = new Context(new ConcreteStrategyB());
      context.ContextInterface();

      context = new Context(new ConcreteStrategyC());
      context.ContextInterface();

      // Wait for user
      Console.ReadKey();
    }
  }

  /// <summary>
  /// The 'Strategy' abstract class
  /// </summary>
  abstract class Strategy
  {
    public abstract void AlgorithmInterface();
  }

  /// <summary>
  /// A 'ConcreteStrategy' class
  /// </summary>
  class ConcreteStrategyA : Strategy
  {
    public override void AlgorithmInterface()
    {
      Console.WriteLine(
        "Called ConcreteStrategyA.AlgorithmInterface()");
    }
  }

  /// <summary>
  /// A 'ConcreteStrategy' class
  /// </summary>
  class ConcreteStrategyB : Strategy
  {
    public override void AlgorithmInterface()
    {
      Console.WriteLine(
        "Called ConcreteStrategyB.AlgorithmInterface()");
    }
  }

  /// <summary>
  /// A 'ConcreteStrategy' class
  /// </summary>
  class ConcreteStrategyC : Strategy
  {
    public override void AlgorithmInterface()
    {
      Console.WriteLine(
        "Called ConcreteStrategyC.AlgorithmInterface()");
    }
  }

  /// <summary>
  /// The 'Context' class
  /// </summary>
```

```
    class Context
    {
      private Strategy _strategy;

      // Constructor
      public Context(Strategy strategy)
      {
        this._strategy = strategy;
      }

      public void ContextInterface()
      {
        _strategy.AlgorithmInterface();
      }
    }
  }



```

Output

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

Real-world code in C#

This real-world code demonstrates the Strategy pattern which encapsulates sorting algorithms in the form of sorting objects. This allows clients to dynamically change sorting strategies including Quicksort, Shellsort, and Mergesort.

```


using System;
using System.Collections.Generic;

namespace DoFactory.GangOfFour.Strategy.RealWorld
{
  /// <summary>
  /// MainApp startup class for Real-World
  /// Strategy Design Pattern.
  /// </summary>
  class MainApp
  {
    /// <summary>
    /// Entry point into console application.
    /// </summary>
    static void Main()
    {
      // Two contexts following different strategies
      SortedList studentRecords = new SortedList();

      studentRecords.Add("Samual");
      studentRecords.Add("Jimmy");
      studentRecords.Add("Sandra");
      studentRecords.Add("Vivek");
      studentRecords.Add("Anna");

      studentRecords.SetSortStrategy(new QuickSort());
      studentRecords.Sort();

      studentRecords.SetSortStrategy(new ShellSort());
      studentRecords.Sort();

      studentRecords.SetSortStrategy(new MergeSort());
      studentRecords.Sort();

      // Wait for user
      Console.ReadKey();
    }
  }

  /// <summary>
  /// The 'Strategy' abstract class
  /// </summary>
  abstract class SortStrategy
  {
    public abstract void Sort(List<string> list);
  }

  /// <summary>
  /// A 'ConcreteStrategy' class
  /// </summary>
  class QuickSort : SortStrategy
  {
    public override void Sort(List<string> list)
    {
      list.Sort(); // Default is Quicksort
      Console.WriteLine("QuickSorted list ");
    }
  }

  /// <summary>
  /// A 'ConcreteStrategy' class
  /// </summary>
  class ShellSort : SortStrategy
  {
    public override void Sort(List<string> list)
    {
      //list.ShellSort(); not-implemented
      Console.WriteLine("ShellSorted list ");
    }
  }

  /// <summary>
  /// A 'ConcreteStrategy' class
  /// </summary>
  class MergeSort : SortStrategy
  {
    public override void Sort(List<string> list)
    {
      //list.MergeSort(); not-implemented
```

```
        Console.WriteLine("MergeSorted list ");
      }
    }

    /// <summary>
    /// The 'Context' class
    /// </summary>
    class SortedList
    {
      private List<string> _list = new List<string>();
      private SortStrategy _sortstrategy;

      public void SetSortStrategy(SortStrategy sortstrategy)
      {
        this._sortstrategy = sortstrategy;
      }

      public void Add(string name)
      {
        _list.Add(name);
      }

      public void Sort()
      {
        _sortstrategy.Sort(_list);

        // Iterate over list and display results
        foreach (string name in _list)
        {
          Console.WriteLine(" " + name);
        }
        Console.WriteLine();
      }
    }
  }



```

Output

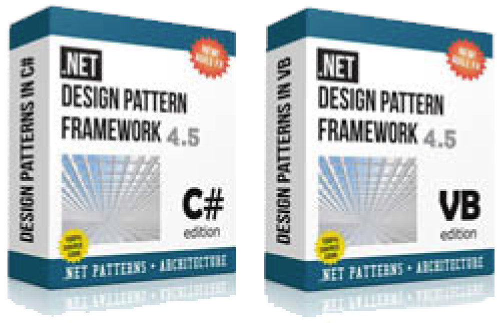
```
QuickSorted list
 Anna
 Jimmy
 Samual
 Sandra
 Vivek

ShellSorted list
 Anna
 Jimmy
 Samual
 Sandra
 Vivek

MergeSorted list
 Anna
 Jimmy
 Samual
 Sandra
 Vivek
```

.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^{TM} 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)