.NET Design Patterns

Design patterns are solutions to software design problems you find again and again in real-world application development. Patterns are about reusable designs and interactions of objects.

The 23 Gang of Four (GoF) patterns are generally considered the foundation for all other patterns. They are categorized in three groups: Creational, Structural, and Behavioral (for a complete list see below).

To give you a head start, the C# source code for each pattern is provided in 2 forms: *structural* and *real-world*. Structural code uses type names as defined in the pattern definition and UML diagrams. Real-world code provides real-world programming situations where you may use these patterns.

A third form, *.NET optimized*, demonstrates design patterns that fully exploit built-in .NET 4.5 features, such as, generics, attributes, delegates, reflection, and more. These and much more are available in our [.NET Design Pattern Framework 4.5](http://www.dofactory.com/products/net-design-pattern-framework). You can see the [Singleton](http://www.dofactory.com/net/singleton-design-pattern) page for a .NET 4.5 Optimized example.

|  |  |
| --- | --- |
| **Creational Patterns** | |
| [Abstract Factory](http://www.dofactory.com/net/abstract-factory-design-pattern) | Creates an instance of several families of classes |
| [Builder](http://www.dofactory.com/net/builder-design-pattern) | Separates object construction from its representation |
| [Factory Method](http://www.dofactory.com/net/factory-method-design-pattern) | Creates an instance of several derived classes |
| [Prototype](http://www.dofactory.com/net/prototype-design-pattern) | A fully initialized instance to be copied or cloned |
| [Singleton](http://www.dofactory.com/net/singleton-design-pattern) | A class of which only a single instance can exist |

|  |  |
| --- | --- |
| **Structural Patterns** | |
| [Adapter](http://www.dofactory.com/net/adapter-design-pattern) | Match interfaces of different classes |
| [Bridge](http://www.dofactory.com/net/bridge-design-pattern) | Separates an object’s interface from its implementation |
| [Composite](http://www.dofactory.com/net/composite-design-pattern) | A tree structure of simple and composite objects |
| [Decorator](http://www.dofactory.com/net/decorator-design-pattern) | Add responsibilities to objects dynamically |
| [Facade](http://www.dofactory.com/net/facade-design-pattern) | A single class that represents an entire subsystem |
| [Flyweight](http://www.dofactory.com/net/flyweight-design-pattern) | A fine-grained instance used for efficient sharing |
| [Proxy](http://www.dofactory.com/net/proxy-design-pattern) | An object representing another object |

|  |  |
| --- | --- |
| **Behavioral Patterns** | |
| [Chain of Resp.](http://www.dofactory.com/net/chain-of-responsibility-design-pattern) | A way of passing a request between a chain of objects |
| [Command](http://www.dofactory.com/net/command-design-pattern) | Encapsulate a command request as an object |
| [Interpreter](http://www.dofactory.com/net/interpreter-design-pattern) | A way to include language elements in a program |
| [Iterator](http://www.dofactory.com/net/iterator-design-pattern) | Sequentially access the elements of a collection |
| [Mediator](http://www.dofactory.com/net/mediator-design-pattern) | Defines simplified communication between classes |
| [Memento](http://www.dofactory.com/net/memento-design-pattern) | Capture and restore an object's internal state |
| [Observer](http://www.dofactory.com/net/observer-design-pattern) | A way of notifying change to a number of classes |
| [State](http://www.dofactory.com/net/state-design-pattern) | Alter an object's behavior when its state changes |
| [Strategy](http://www.dofactory.com/net/strategy-design-pattern) | Encapsulates an algorithm inside a class |
| [Template Method](http://www.dofactory.com/net/template-method-design-pattern) | Defer the exact steps of an algorithm to a subclass |
| [Visitor](http://www.dofactory.com/net/visitor-design-pattern) | Defines a new operation to a class without change |