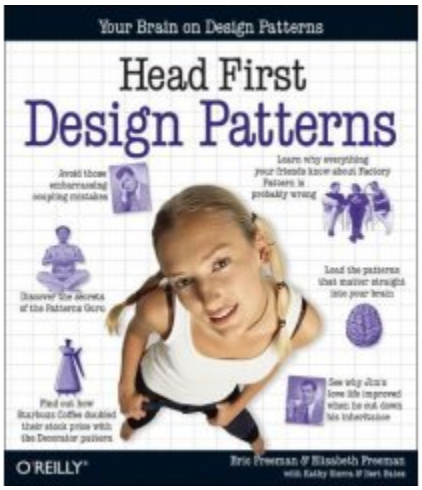
Head First

Design Patterns

for .NET 2.0



Companion document to
Design Pattern Framework™

by

Data & Object Factory
www.dofactory.com

Index

Chapter 1: Intro to Design Patterns

The *Head First Design Patterns* book has taken the developer community by storm and has been a bestseller since the day it was published. What has attracted developers is the whimsical and informal approach to explaining advanced OO concepts and design patterns.

The book comes with a downloadable set of examples in Java. Unfortunately, this poses a challenge for .NET developers because as a developer you don't want to have to deal with language and library differences while studying concepts that are not easy to grasp to begin with.

To alleviate this, the .NET Design Pattern Framework™ from Data & Object Factory includes a complete set of Head First Design Pattern code samples in C# and VB.NET . It includes a total of 46 projects all residing in a single .NET Solution for easy access.

This document does 3 things:

1) it associates the original Java projects with the .NET projects,
2) it references the .NET projects back to the page where the pattern is discussed, and
3) anything noteworthy that came up in the Java - .NET translation it is mentioned here

We hope you will find the .NET code samples useful in your effort to comprehend and learn design patterns.

Chapter 1 has just one coding example: the Strategy pattern.

Page 18: Testing the Duck code

Java program name: strategy

Implemented as DoFactory.HeadFirst.Strategy

Chapter 2: Observer Pattern

Page 57: Implementing the Weather Station

Java program name: observer/WeatherStation

Implemented as DoFactory.HeadFirst.Observer.WeatherStation

This example uses a .NET 2.0 generic List -- List<T> in C# and List(Of T) in VB.NET.

Page 67: Reworking the Weather Station with built-in support

Java program name: observer/WeatherStationObservable

Implemented as DoFactory.HeadFirst.Observer.WeatherStationObservable

.NET does not support the Observer/Observable built-in types so this example uses two hand-coded types: the IObserver interface and the Observable base class. However, a better way to implement the Observer pattern in .NET would be to use .NET multicast delegates as demonstrated in the next example. This code sample uses a .NET 2.0 generic List -- List<T> in C# and List(Of T) in VB.NET.

Page 72: Other places you'll find the Observer Pattern

Java program name: observer/Swing

Implemented as DoFactory.HeadFirst.Observer.DotNet

.NET does not support Swing, therefore this example has been implemented as a console application. In .NET the Observer Pattern is implemented very elegantly as multicast delegates which is demonstrated in this example. To keep the learning experience as close to the original Java code, no .NET 2.0 generic delegates were used. If you'd like to see generic delegates in action the please refer to the .NET optimized example in the Gang of Four Observer pattern.

Chapter 3: Decorator Pattern

Page 95: Writing the Starbuzz Code

Java program name: decorator/starbuzz

Implemented as DoFactory.HeadFirst.Decorator.Starbuzz

Page 100: Real world Decorators: Java (i.e. .NET) I/O

Java program name: decorator/io

Implemented as DoFactory.HeadFirst.Decorator.IO

The IO namespace in .NET uses the Decorator pattern quite extensively. This example demonstrates the use of a CryptoStream which *decorates* a FileStream class. The CryptoStream links data streams to cryptographic transformations (encryption and decryption services). To run this example you need a text file 'MyInFile.txt' with some text in the project directory – for the text in this file you could use "I know the decorator pattern therefore I rule!" as suggested in the Head First Design Patterns book. After you ran the example you'll find 2 new files created in the same directory: one the same as the input file, the other the same but encrypted (using the decorator pattern).

Chapter 4: Factory Pattern

Page 112: Identifying the aspects that vary

Java program name: factory/pizzas

Implemented as DoFactory.HeadFirst.Factory.PizzaShop

This example uses a .NET 2.0 generic List -- List<T> in C# and List(Of T) in VB.NET.

Page 131: It's finally time to meet the Factory Method Pattern

Java program name: factory/pizzafm

Implemented as DoFactory.HeadFirst.Factory.Method.Pizza

Note: page **137** details the DependentPizzaStore which is also available in this project.
The example uses .NET 2.0 generic Lists -- List<T> in C# and List(Of T) in VB.NET.

Page 145: Families of Ingredients…

Java program name: factory/pizzaaf

Implemented as DoFactory.HeadFirst.Factory.Abstract.Pizza

Chapter 5: Singleton Pattern

Page 173: Dissecting the classic Singleton Pattern

Java program name: singleton/classic

Implemented as DoFactory.HeadFirst.Singleton.Classic

Page 175: The Chocolate Factory

Java program name: singleton/chocolate

Implemented as DoFactory.HeadFirst.Singleton.Chocolate

Page 180: Dealing with Multithreading

Java program name: singleton/threadsafe

Implemented as DoFactory.HeadFirst.Singleton.Multithreading

This project includes an EagerSingleton which *eagerly* creates the instance (this occurs when class is loaded for the first time). This provides a robust and thread-safe .NET solution to the multithreading issues discussed in the book.

Page 182: Use “double-checked locking”

Java program name: singleton/dcl

Implemented as DoFactory.HeadFirst.Singleton.DoubleChecked

Chapter 6: Command Pattern

Page 204: Our first command object

Java program name: command/simpleremote

Implemented as DoFactory.HeadFirst.Command.SimpleRemote

Page 210: Implementing the Remote Control

Java program name: command/remote

Implemented as DoFactory.HeadFirst.Command. Remote

Page 216: Undo

Java program name: command/undo

Implemented as DoFactory.HeadFirst.Command.Undo

A .NET enumeration named CeilingFanSpeed was added to replace the HIGH, LOW, MEDIUM, and OFF constants in Java. Java does not support built-in enumerations.

Page 224: Every remote needs a Party Mode!

Java program name: command/party

Implemented as DoFactory.HeadFirst.Command.Party

A .NET enumeration named CeilingFanSpeed was added to replace the HIGH, LOW, MEDIUM, and OFF constants in Java. Java does not support built-in enumerations.

Chapter 7: Adapter and Facade Patterns

Page 238: If it walks like a duck and quacks like a duck…

Java program name: adapter/ducks

Implemented as DoFactory.HeadFirst.Adapter.Duck

Page 249: Adapting an Enumeration to an Iterator

Java program name: adapter/iterenum

Implemented as DoFactory.HeadFirst.Adapter.IterEnum

Unlike Java, there are no legacy Enumeration interfaces in .NET. This example builds on .NET's built-in facility to iterate over different types of collections. C# 2.0 supports even more powerful *iterators* which can iterate over more complex data structures, such as, collections, trees, and graphs. Currently, VB.NET does not support these new iterators. In this example generic collections are used to demonstrate iteration. They are in C#: List<T>, Dictionary<T>, and SortedDictionary<T>, and in VB.NET: List(Of T), Dictionary(Of T) ,and SortedDictionary(Of T).

Page 255: Home Sweet Home Theater

Java program name: facade/hometheater

Implemented as DoFactory.HeadFirst.Facade.HomeTheater

Chapter 8: Template Method Pattern

Page 277: Whipping up some coffee and tea classes (in .NET)

Java program name: template/simplebarista

Implemented as DoFactory.HeadFirst.Template.SimpleBarista

Page 280: Sir, may I abstract your Coffee, Tea?

Java program name: template/barista

Implemented as DoFactory.HeadFirst.Template. Barista

The example also includes the code on page **292**: Hooked on Template Method…

Page 300: Sorting with Template Method

Java program name: template/sort

Implemented as DoFactory.HeadFirst.Template.Sort

Uses the .NET built-in IComparable interface

Page 306: Swinging' with Frames

Java program name: template/frame

Implemented as **DoFactory.HeadFirst.Template.WindowsService**

Swing and Jframe do not exist in .NET. A good example of where Template methods are used in .NET is when you write Windows Services applications. WIndows Services require that you implement several 'hooks' (or Template methods), such as `OnStart()` and `OnStop()`. The Visual Studio.NET generates boilerplate code so that you can simply implement the body of these methods. Note: the example code is a Windows Service and therefore does not run as a standalone executable.

Page 307: Applets

Java program name: template/applet

Implemented as **DoFactory.HeadFirst.Template.Control**

Applets are similar to controls in .NET. This example shows that Windows event handlers are simply 'hooks' that you can choose to implement or not. Most of the time you will implement very few of the many templated events available.

Chapter 9: Iterator and Composite Pattern

Page 317: Menu

Java program name: iterator/dinermerger

Implemented as DoFactory.HeadFirst.Iterator.DinerMerger

Page 327: Reworking Menu with Iterator

Java program name: iterator/dinermergeri

Implemented as DoFactory.HeadFirst..Iterator.DinerMergerI

Page 333: Cleaning things up with java.util.Iterator (.NET Iterator)

Java program name: iterator/dinermergercafe

Implemented as DoFactory.HeadFirst.Iterator.DinerMergerCafe

In following the book, this example uses the built-in.NET IEnumerator interface. However, iterating over collections is far easier using the .NET foreach statement. On page **349** the book talks about iterators and collections in Java 5. It is interesting to note that the new Java 5 `for` statement is just like C#'s `foreach` or VB.NET's `For Each` statement. C# 2.0 enhances iterators even further with the new `'yield return'` and `'yield break'` keywords.

Page 360: Implementing the Menu Component

Java program name: composite/menu

Implemented as DoFactory.HeadFirst.Composite.Menu

This example uses a .NET 2.0 generic List<T> in C# (List(Of T) in VB.NET) to store the collection of MenuComponents.

Page 369: The Composite Iterator

Java program name: composite/menuiterator

Implemented as DoFactory.HeadFirst.Composite.MenuIterator

The .NET implementation was simplified because the Stack iterator example in Java is overly complex. Moreover, it includes (in our view) dubious try/catch usage which also does not add any value to learning Design Pattern concepts.

Chapter 10: State Pattern

Page 388: State Machines 101

Java program name: state/gumball

Implemented as DoFactory.HeadFirst.State.Gumball

An enumeration GumballMachineState replaces the Java contants SOLD_OUT, NO_QUARTER, HAS_QUARTER, and SOLD.

Page 401: Implementing our State classes

Java program name: state/gumballstate

Implemented as DoFactory.HeadFirst.State.GumballState

Page 413: We still need to finish the Gumball 1 in 10 game

Java program name: state/gumballstatewinner

Implemented as DoFactory.HeadFirst.State.GumballStateWinner

Chapter 11: Proxy Pattern

Page 431: Coding the Monitor

Java program name: proxy/gumballmonitor

Implemented as DoFactory.HeadFirst.Proxy.GumballMonitor

Page 451: Getting the GumballMachine ready for remote service

Java program name: proxy/gumball

Implemented as:

DoFactory.HeadFirst.Proxy.GumballState.Client (a console application exe)
DoFactory.HeadFirst.Proxy.GumballState.Server (a console application exe)
DoFactory.HeadFirst.Proxy.GumballState.Machine (a class library dll)

RMI only exists in the Java world. The .NET counterpart is .NET Remoting. In this example we demontrate the use of a .NET Proxy object that is used to invoke a remote class. Three projects are required for this demonstration. Compile the above projects and place the following files in a single directory, say, **c:\test**:

- DoFactory.HeadFirst.Proxy.GumballState.Client.exe
- DoFactory.HeadFirst. Proxy.GumballState.Server.exe
- DoFactory.HeadFirst. Proxy.GumballState.Machine.dll
- ProxyClient.exe.config
- ProxyServer.exe.config

Then open a command box, change directory to **c:\test** and launch the server by typing *DoFactory.HeadFirst.Proxy.GumballState.Server.exe*. Open a second command box, change to **c:\test** and launch the Client by typing *DoFactory.HeadFirst.Proxy.GumballState.Client.exe*. The client GumballMonitor object

will communicate with an instance of the GumballMachine on the Server side using an internal .NET proxy object. The results will be printed on the screen.

Page 462: Get ready for Virtual Proxy

Java program name: proxy/virtualproxy

Implemented as DoFactory.HeadFirst.Proxy.VirtualProxy

This simple .NET Windows Application uses an ImageProxy object. ImageProxy retrieves a book cover image from amazon.com on a separate thread. In the meantime (while retrieving) it provides a placeholder image that is stored locally. Click twice on the button to see Virtual Proxy in action. Note: first of all, you do need Internet access to make it work. Secondly, if the program is not able to retrieve the book cover image from amazon.com, it is most likely related to your firewall configuration.

Page 474: Using .NET API Proxy to create a protection proxy

Java program name: proxy/javaproxy

Implemented as DoFactory.HeadFirst.Proxy.DotNetProxy

A dynamic proxy dynamically generates a class that conforms to a particular interface, proxying all invocations to a single standard method. This functionality is standard in Java. In .NET there are two ways to implement this: one is to use the built-in RealProxy class and another way is to use Reflection.Emit. This example demonstrates the dynamic proxy pattern using the Reflection.Emit method.

The example uses a file named ClassGenerator.cs which was derived from the Open Source Nmock project (nmock.org). The details of this file are rather involved and beyond the scope of this pattern discussion. The VB.NET version of this sample uses the same C# implementation of ClassGenerator (as a .dll).

Chapter 12: Compound Patterns

Page 501: Duck reunion

Java program name: combining/ducks

Implemented as DoFactory.HeadFirst.Combining.Ducks

Page 503: When ducks are around, geese can't be far

Java program name: combining/adapter

Implemented as DoFactory.HeadFirst.Combining.Adapter

Page 506: We're going to make those Quackologists happy

Java program name: combining/decorator

Implemented as DoFactory.HeadFirst.Combining.Decorator

Page 508: We need a factory to produce ducks!

Java program name: combining/factory

Implemented as DoFactory.HeadFirst.Combining.Factory

Page 513: Let's create a flock of ducks

Java program name: combining/composite

Implemented as DoFactory.HeadFirst.Combining.Composite

Page 516: Can you say 'Observer'?

Java program name: combining/observer

Implemented as DoFactory.HeadFirst.Combining.Observer

The sample code makes use of .NET 2.0 generics. Two generic List<T> classes (List(Of T) in VB.NET) are used to maintain collections of IQuackable and IObserver interfaces.

Page 534: Using MVC to control the beat

Java program name: combined/djview

Implemented as DoFactory.HeadFirst.Combined.MVC

Again, Java Swing does not exist in .NET and this example is built as a standalone WinForms application. Instead of Midi a simple timer control is used to generate the beats (each beat includes a Beep). The image on page 530 most closely resembles the implementation in this .NET example. The only exception is line 5 ("I need your state information"); there is no need for the View to query the Model because the state (the BeatsPerMinute) is sent as part of line 4 ("I have changed") using event arguments.

This example also uses a type-safe generic delegate named BeatHandler. The argument named sender is not of type object (which it had to be pre-.NET 2.0) , but rather of type T (i.e. generic and type-safe).