

Laborator 3

Continut:

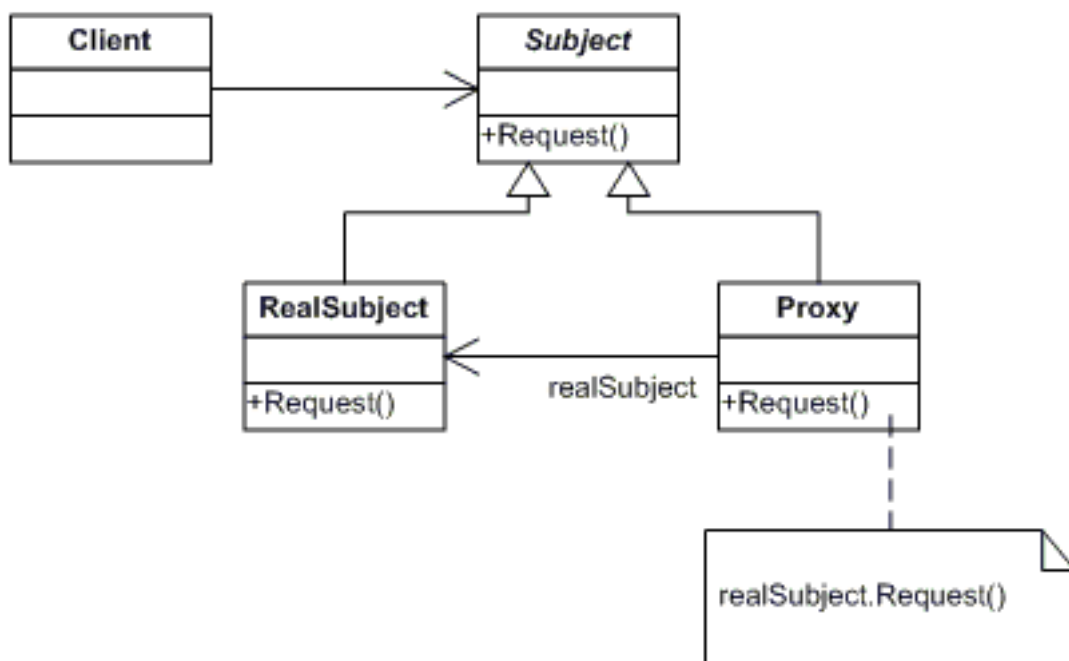
1. Proxy Design Pattern, Lazy loading
2. Microsoft Enterprise Library 6
3. Logging folosind MEL 6

1. Lazy loading via Proxy pattern

1.1. Proxy pattern

Proxy: pattern de design obiectual, prin care se realizeaza o clasa ce se poate folosi ca interfata sau substituent pentru o alta clasa. Proxy-ul este un surogat al clasei reale; operatiile care sunt apelate pe proxy vor fi trimise mai departe catre clasa de destinatie initiala - mai jos: clasa RealSubject, posibil dupa niste operatii suplimentare. Tot aici, e posibil ca dupa apelul catre clasa de destinatie initiala sa se efectueze alte operatii, in cadrul implementarii din clasa Proxy.

Diagrama [1]:



Cod:

a. C#

// Proxy pattern - Structural example

```
using System;
namespace Structural
{
    /// <summary>
```

```

/// MainApp startup class for Structural
/// Proxy Design Pattern.
/// </summary>
class MainApp
{
    /// <summary>
    /// Entry point into console application.
    /// </summary>
    static void Main()
    {
        // Create proxy and request a service
        ISubject proxy = new Proxy();
        proxy.Request();

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

/// <summary>
/// The 'ISubject' interface
/// </summary>
interface ISubject
{
    void Request();
}

/// <summary>
/// The 'RealSubject' class
/// </summary>
class RealSubject : ISubject
{
    public void Request()
    {
        Console.WriteLine("Called RealSubject.Request()");
        //real work goes here
    }
}

/// <summary>
/// The 'Proxy' class
/// </summary>
class Proxy : ISubject
{
    private RealSubject _realSubject;

    public override void Request()
    {
        //here: add any other step that must be accomplished BEFORE the call: security checking, logging, etc.
        // Use 'lazy initialization'
        if (_realSubject == null)
        {

```

```

    _realSubject = new RealSubject();//create the real worker class
}
_realSubject.Request();//forwards call to the worker class
//here: add any other step that must be accomplished AFTER the call
}
}
}

```

b. Java:

```
import java.util.*;
```

```
interface Image {
    public void displayImage();
}

```

//on System A

```
class ReallImage implements Image {
    private String filename;
    public ReallImage(String filename) {
        this.filename = filename;
        loadImageFromDisk();
    }

    private void loadImageFromDisk() {
        System.out.println("Loading " + filename);
    }

    public void displayImage() {
        System.out.println("Displaying " + filename);
    }
}

```

//on System B

```
class ProxyImage implements Image {
    private String filename;
    private Image image;

    public ProxyImage(String filename) {
        this.filename = filename;
    }
    public void displayImage() {
        image = new ReallImage(filename);
        image.displayImage();
    }
}

```

```
class ProxyExample {
    public static void main(String[] args) {
        Image image1 = new ProxyImage("HiRes_10MB_Photo1");
        Image image2 = new ProxyImage("HiRes_10MB_Photo2");
    }
}

```

```

    image1.displayImage(); // loading necessary
    image2.displayImage(); // loading necessary
}
}

```

Observatii: un proxy:

- Mentine o referinta la un obiect de tipul interfetei; prin referinta se face acces la obiectul real;
- Pune la dispozitie o interfata identica cu a obiectului pe care il proxifica; de exemplu, implementeaza aceeasi interfata ca si obiectul real – varianta cea mai comuna
- Controleaza accesul la obiectul real si este responsabil de crearea si stergerea lui
- Poate implementa alte responsabilitati:
 - Remote proxy: codifica cererea si o trimite catre un alt process/alta masina; responsabilitatea proxy-ului este facilitarea comunicarii catre obiectul real (creare de socket, serializare etc)
 - Virtual proxy: poate face caching pentru obiectul real sau pentru date pe care acesta le-a mai calculat anterior; cererile ulterioare pot beneficia de stocarea datelor in cache
 - Protection proxy: verifica daca sunt indeplinite drepturile de acces (securitate, autorizare)

1.2. Lazy loading

Prin Lazy loading se implementeaza incarcarea datelor doar in momentul in care este efectiv nevoie de ele. De exemplu, pentru o lista de item-uri care se vrea a fi incarcata din baza de date, se poate amana momentul incarcarii. Beneficiul este ca se evita incarcarea devreme (= eager loading, opusul lui lazy loading) a unui continut care este posibil sa nu fie de fapt utilizat niciodata. Dezavantajul este ca poate duce la marirea numarului de apeluri la distanta (un apel pentru aducerea obiectului continator plecand de la ce e stocat in baza de date, inca un alt apel pentru lazy loading). Este un compromis, uneori benefic pentru aplicatie.

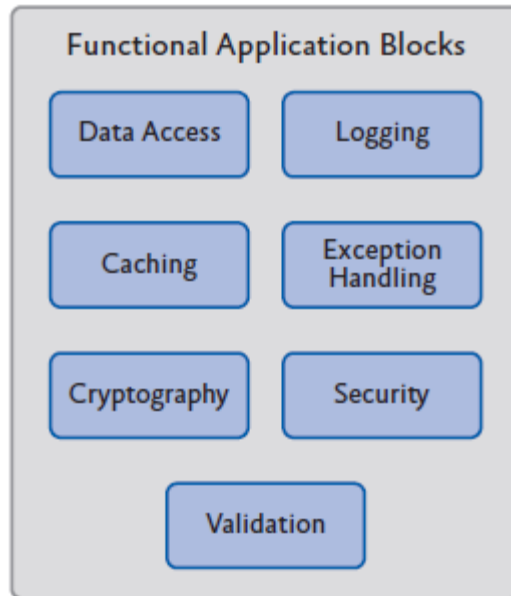
Exemplu de implementare: proiectul “Design Pattern Framework 2.0 CS”, directorul “Patterns In Action”, clasa `ProxyList<T>` – proxy pentru lista de `Order(s)` asociata unui client sau pentru `OrderDetails` asociata unui `Order`. Urmariti implementarile din fisierele `BusinessObjects/ProxyList.cs`, `Facade/ProxyObjects/ ProxyForOrders.cs`, `Facade/ProxyObjects/ ProxyForOrderDetails.cs`.

2. Microsoft Enterprise Library

MEL 6.0 – functioneaza pe .NET framework 4.5+; cerintele tehnice sunt date la <https://www.microsoft.com/en-us/download/details.aspx?id=15104> sunt: “Operating systems: Microsoft Windows® 8, Microsoft Windows® 7, Windows Server 2008 R2, Windows Server 2012.” la care desigur se adauga si alte versiuni ulterioare de sisteme de operare Windows.

Resursa: [3]

Blocuri:



3. Logging (jurnalizare)

Logging: înregistrarea evenimentelor de interes obținute pe parcursul rularii unei aplicații.

Tipuri de logging:

- instrumentation: logging pentru urmărirea “stării de sănătate” a aplicației;
- tracing (urmărire) și debugging (eliminarea erorilor de programare): include informații relativ la ce s-a întâmplat la un punct anume în aplicație, precum și starea din acel timp. Exemplu: procesarea unor date se poate face doar dacă acestea au o stare convenabilă (toate câmpurile necesare umplute cu date coerente, coerenta stării curente); dacă la un moment dat se întâlnește o dată (situație) care nu satisface cerințele, atunci se înregistrează evenimentul împreună cu cât mai multe detalii. Ulterior, fișierul jurnal este cercetat pentru a se deduce cum anume s-a ajuns la situația creată și dacă asta necesită intervenție pe cod sau nu.
- logging de audit – monitorizarea accesului la date; menținerea unui istoric al prelucrărilor efectuate asupra unor date de importanță mai mare (e.g. date confidențiale); se pot astfel detecta schimbările efectuate asupra unor înregistrări, eventual se poate reveni la valorile vechi.

Există Logging Application Block care permite utilizarea oricărui tip de logging din cele descrise mai sus.

Mesajele de log pot fi salvate în:

- jurnalul de mesaje menținut de sistemul de operare
- mesaj email
- baza de date
- coada de mesaje (MSMQ)
- fișier text pe disc
- eveniment WMI
- locații personalizate folosind extensii de blocuri de aplicație

Dependinte (referinte de adaugat): Microsoft.Practices.EnterpriseLibrary.Logging.dll, Microsoft.Practices.EnterpriseLibrary.Common.dll din directorul in care s-a instalat pachetul de EnterpriseLibrary. Se recomanda si adaugarea de referinte la Microsoft.Practices.Unity.dll si Microsoft.Practices.Unity.Interception.dll. *In cazul in care se face log in baza de date, e nevoie si de Microsoft.Practices.EnterpriseLibrary.Logging.Database.dll + Microsoft.Practices.EnterpriseLibrary.Data.dll.*

Alternativ, se poate folosi NuGet (vedeti mai jos)

Fragment de cod:

```
if (Logger.IsLoggingEnabled())
{
    LogEntry logEntry = new LogEntry()
    {
        EventId = 100,
        Priority = 2,
        Message = "mesaj de informare",
        Severity = System.Diagnostics.TraceEventType.Information
    };

    logEntry.ExtendedProperties["locatie"] = this.GetType().ToString();

    Logger.Write(logEntry);
}
```

Nota: in MEL versiunea 6 este nevoie ca sa se seteze un obiect de tip **LogWriter** inainte de a se cere jurnalizare. Se poate executa la inceputul aplicatiei codul:

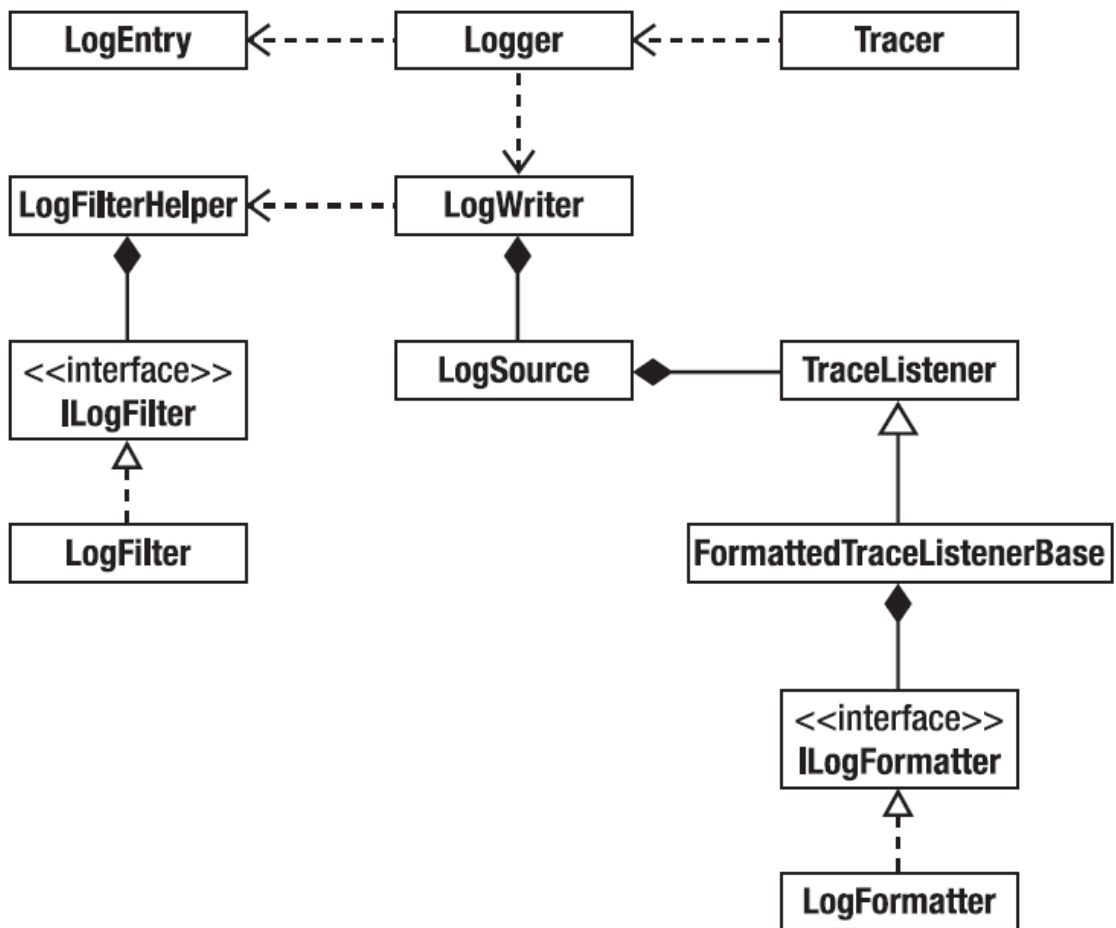
```
Logger.SetLogWriter(new LogWriterFactory().Create());
```

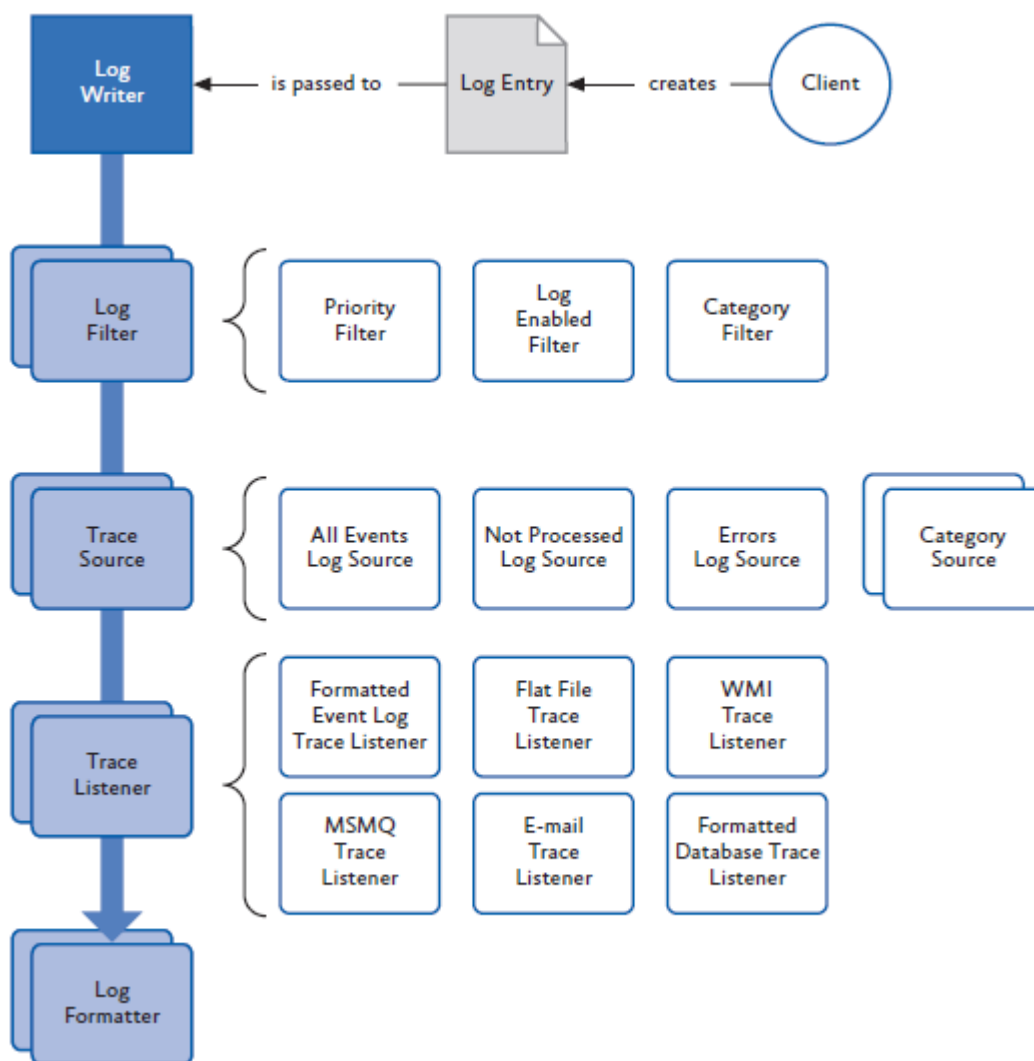
Sursa: <http://entlib.codeplex.com/discussions/442089>

Clase:

- **LogEntry:** reprezinta o intrare ce va fi inregistrata in jurnal. Logging application block seteaza automat id-ul procesului, numele de process, id-ul thread-ului. Programatorul trebuie sa specifice valori pentru mesaj, titlu, prioritate, severitate, eventId.
- **LogFormatter** – de regula un TextFormatter sau BinaryFormatter. Rolul formatorilor in Logging Application Block este de a specifica modul in care se doreste sa arate mesajele jurnalizate. In cea mai mare parte a timpului se va folosi TextFormatter, cu o proprietate Template ce specifica modul in care arata mesajul. In cadrul acestui template pot exista niste specificatori predefiniti (e.g. {timestamp}) pe care Logging Application Block ii va umple cu valori adecvate.
- **TraceListener** – furnizeaza serviciile de jurnalizare. Ei reprezinta calea pe care o urmeaza mesajele in drumul spre destinatie (baze de date, email, fisier, log de evenimente). Logging Application Block vine cu cateva implementari de trace listeners: Database TraceListener, Email TraceListener, Flat File TraceListener, Formatter Event Log TraceListener, etc. TraceListener cere un obiect de tip Formatter pentru a specifica formatul mesajelor
- **LogSource** – o colectie de TraceListener.
- **LogWriter** – piesa centrala. Uneste obiectele de tip tracelister, filter si tot restul pentru a se asigura ca mesajele jurnalizate ajung unde trebuie. Metoda care face acest lucru este LogWriter.Write(LogEntry log).

Diagrama:





Fluxul de lucru:

Biblioteca de Microsoft Enterprise Library (MEL) pentru logging poate fi accesata in doua moduri:

1. (recomandat) Se descarca prin NuGet, specificandu-se pachetul EnterpriseLibrary.Logger
2. Se descarca MEL de la <http://www.microsoft.com/en-us/download/details.aspx?id=38789>

Configurarea se face cel mai simplu prin Enterprise Library config console, care se obtine din:

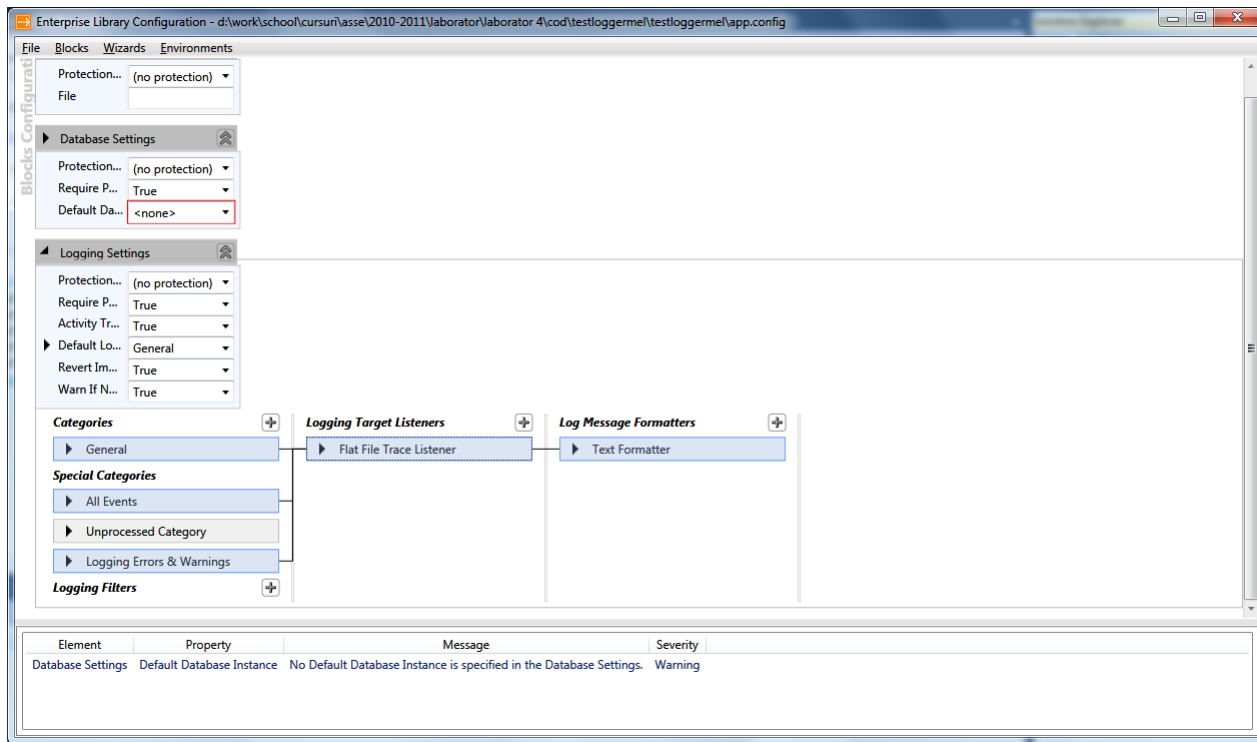
1. Kitul de MEL, daca s-a urmat punctul 1 de mai sus, sau
2. Se foloseste executabilul EntLibConfig.exe **din directorul tools**; va folosi fisierul Microsoft.Practices.EnterpriseLibrary.Logging.dll din acelasi director pentru a putea adauga o sectiune de configurare pentru logging.

Proof of concept:

Se creeaza o aplicatie Windows/WPF, se adauga sau se foloseste fisierul de configurare app.config; deschiderea se poate face cu click dreapta -> Edit Enterprise Library Configuration (instalat odata cu MEL sau folosind VSIX)

Alternativa:

Lansare Enterprise Library Configuration din directorul tools, apoi deschidere de fisier app.config din directorul aplicatiei, adaugare de Logging application block:



Nota: pentru a putea folosi blocurile din MEL, acestea trebuie sa fie prezenta sub forma de fisiere dll in directorul din care se lanseaza consola de configurare!

Pentru demonstratie la laborator, loggerul va fi setat ca un FlatFile Trace Listener.

Tracing:

```
using (new Tracer("activitate cu durata urmarita"))
{
    //aici sunt instructiuni pentru care se doreste urmarirea
    momentului de inceput si de sfarsit
    Thread.Sleep(1000);
}
```

Urmărirea timpului scurs se face pe baza unui identificator unic ([GUID](#)) create de catre sistem.

Download de biblioteca: <http://www.microsoft.com/downloads/details.aspx?familyid=1643758B-2986-47F7-B529-3E41584B6CE5&displaylang=en>

Documentatie: <http://msdn.microsoft.com/en-us/library/cc309506.aspx> (tutorial)

The Definitive Guide to the Microsoft Enterprise Library (detaliat)

Teme de lucru la laborator:

1. Instalati Microsoft Enterprise Library versiunea 6 sau descarcati pachetul prin NuGet
2. Faceti log in baza de date SQL Server folosind MEL 6. Scriptul SQL de creare al BD si exemple se gasesc in codul sursa (fisierul EnterpriseLibrary6-source.exe), directorul Blocks\Logging\Src\DatabaseTraceListener\Scripts
3. Configurati un listener care sa trimita emailuri pentru logurile de tip error.

Bibliografie

- [1] <http://www.dofactory.com/patterns/patternproxy.aspx>
- [2] http://en.wikipedia.org/wiki/Proxy_pattern
- [3] Developer's Guide to Microsoft Enterprise Library
- [4] <http://entlib.codeplex.com/> : hands-on labs, webcasts, sample files