**Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente**

**Parte 3: Excepciones de la librería estándar**

**Briguera, Carivali, Guerrero, Juáres**

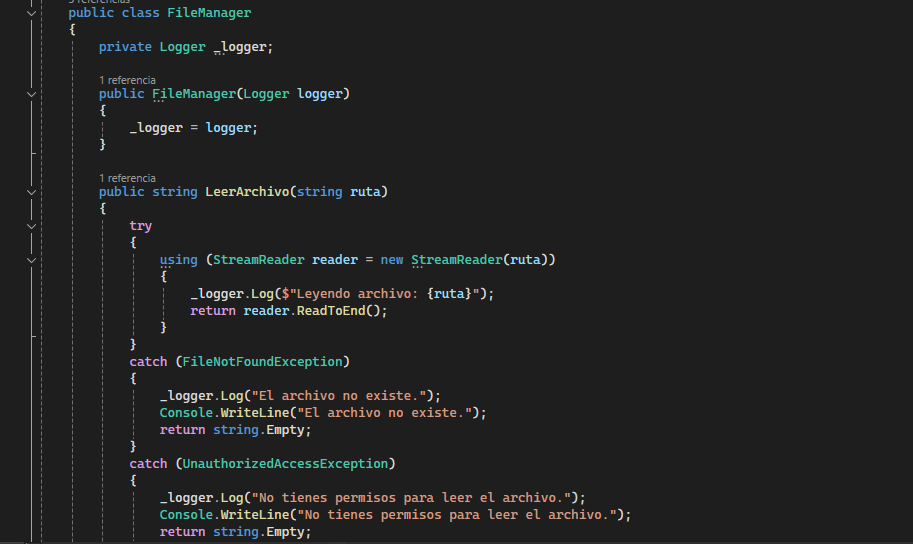
**Universidad Blas Pascal**

**Programación genérica y eventos**

En este documento se desarrollará la integración de gestión de recursos y manejo de excepciones en una aplicación completa, con una investigación de profundización sobre un tema relacionado.

1. **Integración de Funcionalidades:**
   * Combinar las clases FileManager y Calculator en una sola aplicación de consola que permita al usuario realizar operaciones de archivo y cálculos, con manejo de excepciones adecuado.

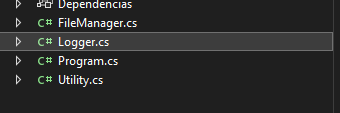
Ejemplo de implementación:

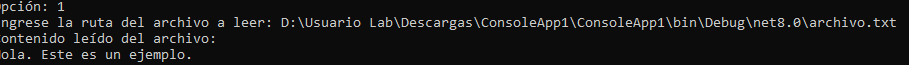


* + Implementar un menú de opciones para que el usuario pueda seleccionar qué operación desea realizar.

Ejemplo de implementación:

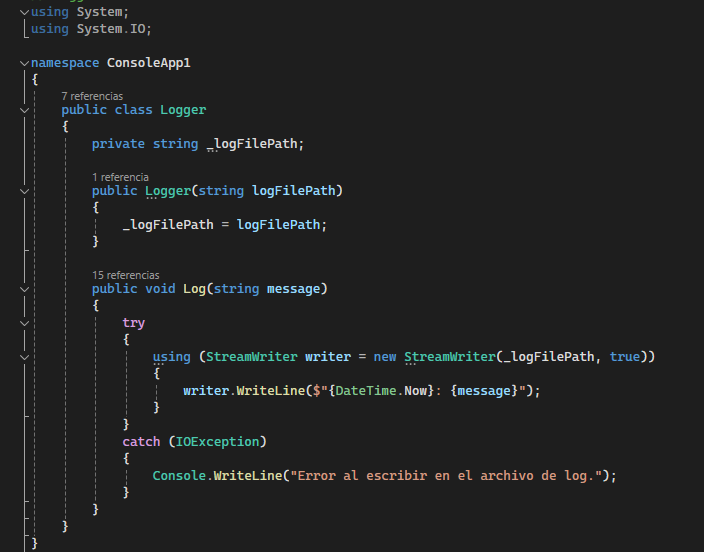


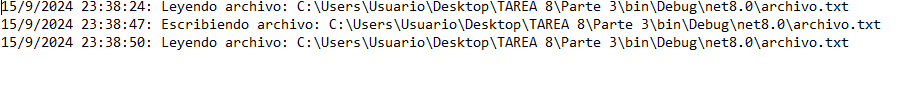




1. **Implementación de Características Avanzadas:**
   * Añadir un sistema de logging que registre todas las excepciones y operaciones realizadas en un archivo de log.

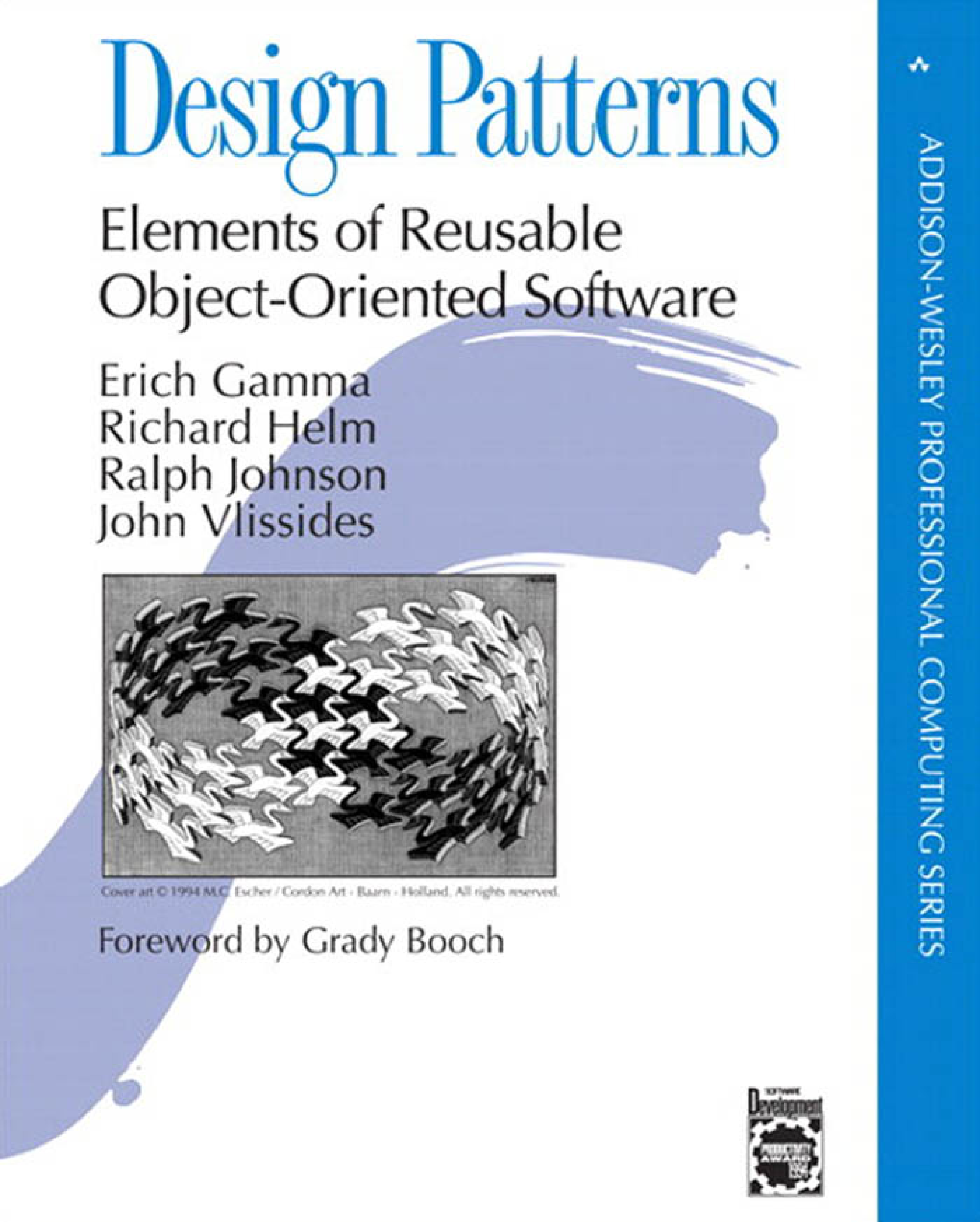
Ejemplo de implementación:





1. **Investigación de Profundización:**

Investigación



Singleton es un patrón de diseño del tipo creacional cuyo propósito es garantizar la existencia de una sola instancia de una clase. Además el acceso a esa única instancia tiene que ser global.

public sealed class Singleton

{

    private readonly static Singleton \_instance = new Singleton();

    private Singleton()

    {

    }

    public static Singleton Instance

    {

        get

        {

            return \_instance;

        }

    }

}

Algunas claves para crear este patrón:

1. **Constructor privado**
2. **Campo estático privado** con inicialización directa (inline)
3. **Propiedad estática pública** que devuelve el campo instanciado

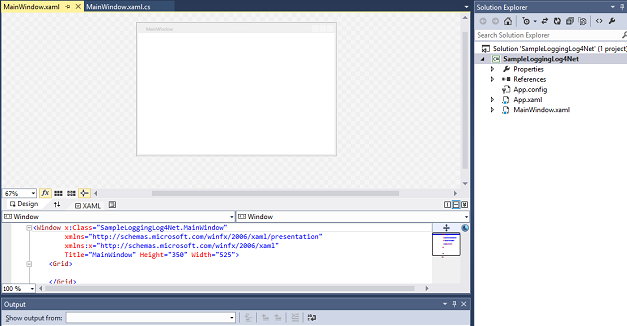
**En Loggers**. Es bueno utilizar singleton porque principalmente su existencia no afecta al resultado de la aplicación. Tanto si el Logger está activado como desactivado el resultado no varía. En cambio en otras aplicaciones puede llegar a significar un problema dado sus aspectos negativos tales como: **Esconden las dependencias, violan el principio de responsabilidad única.** Implícitamente **generan código acoplado**

**Log4Net**

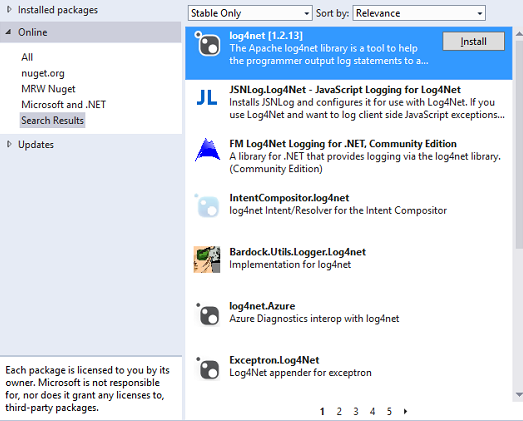
 es una librería licenciada bajo la [Apache Software Licence](http://www.apache.org/licenses/LICENSE-2.0), que nos permite incluir en nuestra aplicación la funcionalidad de “logging” de una forma rápida y sencilla  Podemos descargar la librería directamente desde su [página](http://logging.apache.org/log4net/download_log4net.cgi) o bien instalarla directamente en nuestra aplicación mediante nuget.

Ejemplo de implementación

 ejemplo sencillo utilizando **Windows Presentation Foundation** (WPF).

Lo primero es crear el proyecto 

Seguidamente abrimos el gestor de paquetes de Nuget y buscamos Log4Net.



Seguidamente abrimos el gestor de paquetes de Nuget y buscamos Log4Net.

public ILog Logger { get; set; }

public MainWindow()

{

InitializeComponent();

this.Logger = LogManager.GetLogger(Assembly.GetExecutingAssembly().GetTypes().First());

log4net.Config.XmlConfigurator.Configure();

this.Logger.Info("Inicio de la app");

}

Creamos una propiedad Logger que inicializamos para obtener una instancia que implementa la interfaz **ILog finalmente:** configuramos log4Net mediante XmlConfigurator.

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />

</configSections>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

</startup>

<log4net>

<root>

<level value="ALL"/>

<appender-ref ref="FileAppender" />

</root>

<appender name="FileAppender" type="log4net.Appender.FileAppender">

<file value="log-file.txt" />

<appendToFile value="true" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date [%thread] %-5level %logger [%property{NDC}] - %message%newline" />

</layout>

</appender>

</log4net>

</configuration>

Luego le decimos el nivel de severidad que vamos a registrar:

<root>

<level value="ALL"/>

<appender-ref ref="FileAppender" />

</root>

Log4Net tiene diferentes tipos de “appenders” .Almacenamiento en base de datos (SQL Server, Oracle, SQLite, …), Asp.Net Appender, Buffer de almacenamiento antes de su volcado, consola, consola con distinción de colores, EventLog del sistema, NetSend, RolingFile, Smtp, son algunos de la gran variedad disponible de tipos de “Appenders”.

Un appender es el encargado de procesar un mensaje de log enviado por la aplicación. Usualmente se encarga de almacenarlo en algún medio persistente.

En nuestro ejemplo:

<appender name="FileAppender" type="log4net.Appender.FileAppender">

<file value="Logs\log-file.txt" />

<appendToFile value="true" />

<layout type="log4net.Layout.PatternLayout">

<conversionPattern value="%date [%thread] %-5level %logger [%property{NDC}] - %message%newline" />

</layout>

</appender>

Usar una librería de logging ofrece ventajas sobre una implementación casera. Son más confiables, ya que están probadas y utilizadas por muchos. También proporcionan funcionalidades avanzadas, como diferentes niveles de logging y formatos de salida. Permiten hacer ajustes sin cambiar el código. Además, facilitan el manejo de múltiples destinos de salida, como archivos y consolas, lo que simplifica la gestión de logs en aplicaciones complejas. Básicamente es mejor porque ahorran tiempo y esfuerzo, además de mejorar la calidad del registro de eventos.