|  |
| --- |
| Kaspersky LAB |
| Разработчик ПО |
| Тестовое задание |
|  |
|  |
|  |

|  |
| --- |
|  |

# Задача 1

Есть ли в реализации класса какие-то проблемы и недостатки? Если есть, то перечислите какие и предложите изменения:

public class Processor

{

public void ProcessFile(string filename)

{

Stream fileStream = File.OpenRead(filename);

Console.WriteLine(ReadAllContent(fileStream));

fileStream.Close();

}

public string ReadAllContent(Stream stream)

{

StreamReader streamReader = new StreamReader(stream);

return streamReader.ReadToEnd();

}

}

Да, тут не предусмотрено освобождение файла в случае возникновения исключения. Я бы написал этот класс так:

public class Processor

{

public void ProcessFile(string filename)

{

using (Stream fileStream = File.OpenRead(filename))

{

Console.WriteLine(ReadAllContent(fileStream));

}

}

public string ReadAllContent(Stream stream)

{

using (StreamReader streamReader = new StreamReader(stream))

{

return streamReader.ReadToEnd();

}

}

}

Так как тут выполняется закрытие StreamReader, то после вызова метод ReadAllContent продолжать работу с потоком будет нельзя. Этот метод публичный, так что в нем можно не закрывать StreamReader, если создатели потоков будут заботиться об их закрытии.

# Задача 2

Будет ли работать реализация GetFileHandler в многопоточной среде. Поясните свой ответ

public class FileHandler

{

public static Stream GetFileHandler()

{

if (\_file!=null)

return \_file;

lock(\_lock)

{

if (\_file != null)

return \_file;

\_file = File.OpenRead("name");

return \_file;

}

}

private static object \_lock = new object();

private static Stream \_file;

}

Для полной реализации паттерна Singleton тут не хватает только приватного конструктора и объявления скрытого поля \_file как volatile. Соответственно возможен случай, когда доступ к полю \_file будет получен до того, как оно будет полностью проинициализировано. Работа с самим потоком в многопоточной среде не безопасна и требует дальнейшей синхронизации.

# Задача 3

Есть класс, который занимается обработкой файлов разного формата. Предполагается, что будут добавляться новые форматы файлов. Какие вы видите проблемы в реализации.

Проанализируйте реализацию данного класса и предложите рефакторинг, позволяющий расширять форматы обрабатываемых файлов.

public class FileProcessor

{

enum FileType

{

Html,

Text

}

public void ProcessFile(string fileName)

{

StreamReader fileStream = new StreamReader(File.OpenRead(fileName));

string fileContent = fileStream.ReadToEnd();

if (fileContent.IndexOf("<html") != -1)

ProcessHtmlFile(fileContent);

else ProcessTextFile(fileContent);

ProcessFile(fileContent,fileContent.IndexOf("<html")!

= -1?FileType.Html:FileType.Text);

fileStream.Close();

}

private void ProcessFile(string content, FileType fileType)

{

switch(fileType)

{

case FileType.Html:

ProcessHtmlFile(content);

break;

case FileType.Text:

ProcessTextFile(content);

break;

default:

throw new Exception("Unknown file format");

}

}

private void ProcessHtmlFile(string content)

{

....

}

private void ProcessTextFile(string content)

{

....

}

}

Во первых в данном примере не освобождается файл в случае ошибки. Методы обработки фалов я бы инкапсулировал в отдельных классах, а алгоритм определения метода обработки файла я бы вынес в фабрику, что позволит создавать дополнительные классы обработчики не изменяя текущий. Я бы сделал так, что позволит добавлять и методы определения типа фалов не изменяя существующие:

[AttributeUsage(AttributeTargets.Property)]

public abstract class FileTypeCheckerAttribute : Attribute

{

public abstract bool Check(string fileName, string content);

}

[AttributeUsage(AttributeTargets.Field, AllowMultiple = true)]

public class ContainsAttribute : FileTypeCheckerAttribute

{

public string Text

{

get;

private set;

}

public ContainsAttribute(string text)

{

Text = text;

}

public override bool Check(string fileName, string content)

{

return content.IndexOf(Text) != -1;

}

}

[AttributeUsage(AttributeTargets.Field, AllowMultiple = false)]

public class FileProcessorAttribute : Attribute

{

public Type FileProcessor

{

get;

private set;

}

public FileProcessorAttribute(Type fileProcessor)

{

if(fileProcessor.GetInterface("IFileProcessor") == null)

throw new ArgumentException("The fileProcessor must implement the IFileProcessor interface.");

FileProcessor = fileProcessor;

}

}

public enum FileType

{

[FileProcessor(typeof(HtmlFileProcessor))]

[Contains("<html")]

Html,

[FileProcessor(typeof(TextFileProcessor))]

Text

}

public interface IFileProcessor

{

void ProcessFile(string content);

}

public class TextFileProcessor : IFileProcessor

{

public void ProcessFile(string content)

{

Console.WriteLine("Text: " + content);

}

}

public class HtmlFileProcessor : IFileProcessor

{

public void ProcessFile(string content)

{

Console.WriteLine("Html: " + content);

}

}

public abstract class BaseFileProcessorFactory

{

public abstract IFileProcessor GetFileProcessor();

}

public class FileProcessorFactory : BaseFileProcessorFactory

{

public string FileName

{

get;

private set;

}

public string Content

{

get;

private set;

}

public FileProcessorFactory(string fileName, string content)

{

FileName = fileName;

Content = content;

}

public FileType GetFileType()

{

foreach (FileType fileType in Enum.GetValues(typeof(FileType)))

{

IEnumerable<FileTypeCheckerAttribute> attributes = GetAttributes<FileTypeCheckerAttribute>(fileType);

bool flag = true;

foreach (var attribute in attributes)

{

if (!attribute.Check(FileName, Content))

{

flag = false;

break;

}

}

if (flag)

return fileType;

}

//default

return FileType.Text;

}

public override IFileProcessor GetFileProcessor()

{

FileType fileType = GetFileType();

IEnumerable<FileProcessorAttribute> attributes = GetAttributes<FileProcessorAttribute>(fileType);

if(!attributes.Any())

throw new ArgumentOutOfRangeException("No processor for this file format.");

return Activator.CreateInstance(attributes.First().FileProcessor) as IFileProcessor;

}

private IEnumerable<T> GetAttributes<T>(FileType fileType) where T:Attribute

{

Type type = typeof(FileType);

return (IEnumerable<T>)Attribute.GetCustomAttributes(type.GetField(Enum.GetName(type, fileType)), typeof(T));

}

}

public class FileProcessor

{

public void ProcessFile(string fileName)

{

string fileContent;

using (StreamReader fileStream = new StreamReader(File.OpenRead(fileName)))

fileContent = fileStream.ReadToEnd();

BaseFileProcessorFactory fileProcessorFactory = new FileProcessorFactory(fileName, fileContent);

fileProcessorFactory.GetFileProcessor().ProcessFile(fileContent);

}

}

# Задача 4

Есть клиент, который считывает сообщения из сети по TCP/IP. Сообщения представляют собой XML и бывают трех типов (A,B,C).Клиент занимается обработкой сообщений. В процессе обработки сообщений происходит их разбор и сохранение в постоянном файловом хранилище. Кроме того, подсчитывается статистика по обработке сообщений. Предложите свой вариант дизайна реализации(достаточно показать идею).

Само приложение я бы разбил приложение на следующие уровни.

1. Транспортный уровень - работа с сокетами, обработка всех исключительных ситуаций чтения данных из сокетов, отдает Stream.
2. Уровень чтения xml - читает xml из потока, определяет тип сообщения и читает общие для всех сообщений данные. Используя абстрактную фабрику, создает объект для разбора конкретного типа сообщений и передает ему дальнейшую обработку xml. Отвечает за выявление общих ошибок формата xml.
3. Уровень разбора конкретного типа сообщения – читает данные специфичные для данного типа сообщения и отвечает за специфичные для данного типа сообщения ошибки фомата xml. Отдает объект сообщения, реализующий общий для всех типов сообщений интерфейс.
4. Уровень сериализации сообщений - сохраняет сообщения в файловое хранилище в нужном формате.
5. Уровень обработки сообщений - в зависимости от типа сообщения создается нужный обработчик сообщения, который принимает объект сообщения и выдает объект сбора статистики.
6. Уровень статистики – обрабатывает объекты сбора статистики каждого сообщения.

В качестве платформы можно использовать WCF, тогда возможно потребуется разработать свою привязку.

# Задача 5

Напишите метод, позволяющий посчитать количество элементов в коллекции, удовлетворяющих предикату. Предикат должен передаваться функции параметром.

public int Count<T>(ICollection<T> collection, Func<T, bool> predicate)

{

return collection.Count(predicate);

}

Или так без расширяющих методов:

public static int Count<T>(ICollection<T> collection, Func<T, bool> predicate)

{

int count = 0;

foreach (T item in collection)

{

if(predicate(item))

count++;

}

return count;

}

# Задача 6

Напишите метод, расширяющий BitArray. Метод должен инициализировать созданный объект BitArray целым числом (int).

public static void Initialize(this BitArray array, int value)

{

array.Length = 32;

int multiplier = 1;

for (int i = 0; i < 32; i ++)

{

array.Set(i, (multiplier & value) > 0);

multiplier \*= 2;

}

}

Или так, если знать что он внутри хранит массив int☺

public static void InitializeHack(this BitArray array, int value)

{

array.Length = 32;

FieldInfo fi = typeof(BitArray).GetField("m\_array", BindingFlags.NonPublic | BindingFlags.Instance);

if (fi == null)

throw new Exception("Framework is unknown.");

fi.SetValue(array, new[] { value });

}