# Врезка. Потоки ввода/вывода в .NET

Потоки ввода/вывода - это одна из областей, в которой отчетливо видно влияние паттернов проектирования. Я довольно длительное время постоянно путался в обязанностях между классами TextReader, StreamReader, StringReader и другими. Если у вас была схожая проблема, то этот раздел позволит расставить все по своим местам.

Чтобы понять обязанности классов пространства имен System.IO, нужно нарисовать диаграмму с ключевыми классами и их отношениями (рис. 3.5.1).

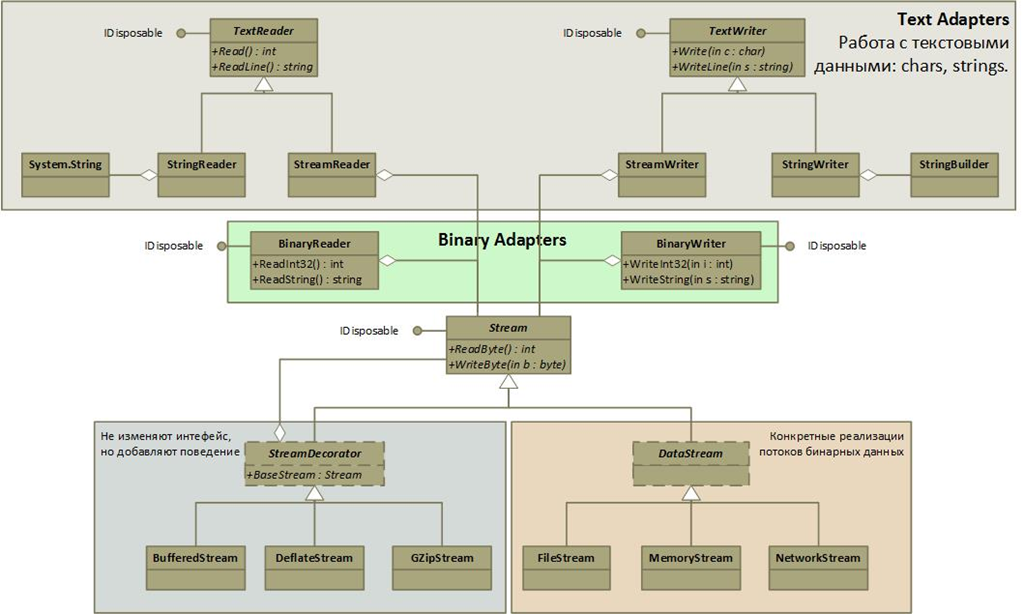


Рисунок 3.5.1 - Классы пространства имен System.IO

Потоки ввода/вывода используют следующие ключевые паттерны.

* Абстракция потоков ввода/вывода: классы System.IO.Stream и наследники - FileStream, MemoryStream, NetworkStream и другие. "Стратегия" потоков позволяет абстрагироваться от конкретной реализации и работать с разными потоками ввода/вывода унифицированным образом.
* Декораторы (BufferedStream, GZipStream и другие). Добавляют определенное поведение потокам ввода/вывода, типа буферизации, сжатия и т.п. Многие другие библиотеки, такие как WCF, добавляют свои декораторы для собственных нужд.
* Адаптеры (текстовые - TextReader/TextWriter и бинарные - BinaryReader/BinaryWriter). Упрощают чтение специфических данных из любых потоков ввода/вывода, например, чтение текстовых данных или примитивных данных из двоичного потока.
* Фасады и фабрики. Класс File является фасадом, который упрощает работу с файлами. Он же содержит ряд фабричных методов для открытия файлов в текстовом/двоичном режимах, для добавления данных или создания нового файла (File.OpenRead, File.OpenWrite, File.Create, File.AppendText, File.OpenText и другие).

## Встроенная поддержка тестируемости

Библиотека потоков ввода/вывода содержит множество абстракций, которые позволяют сделать решение тестируемым. Классы TextReader/TextWriter отлично подходят для этих целей, поскольку помимо классов StreamReader/StreamWriter, которые оперируют потоками ввода/вывода, существуют классы StringReader/StringWriter, которые оперируют строками.

Это значит, что для обеспечения тестируемости, классам достаточно работать не с потоками ввода/вывода, а с классами TextReader/TextWriter, и не изобретать свои собственные абстракции (листинг 3.5.1) (\*):

(\*) Сноска: пример тестируемости классов на основе StreamReader уже был рассмотрен в главе о паттерне Фабричный Метод. Здесь же пример дублируется в несколько измененном виде для наглядности.

public class LogEntryParser  
{  
 private readonly TextReader \_reader;  
  
 public LogEntryParser(TextReader reader)  
 {  
 \_reader = reader;  
 }  
  
 public IEnumerable<LogEntry> Parse()  
 {  
 string line;  
 while ((line = \_reader.ReadLine()) != null)  
 {  
 yield return ParseLine(line);  
 }  
 }  
  
 private LogEntry ParseLine(string line)  
 {  
 return new SimpleLogEntry();  
 }  
}

Листинг 3.5.1 - Класс LogEntryParser

В тесте достаточно создать строку с нужным содержимым, и затем передать ее конструктору StringReader (листинг 3.5.2):

[TestFixture]  
class LogEntryParserTests  
{  
 [Test]  
 public void OneLineSequenceProducesOneItem()  
 {  
 var sr = new StringReader("2015-01-19 [INFO] Message");  
 var cut = new LogEntryParser(sr);  
  
 Assert.That(cut.Parse().Count(), Is.EqualTo(1));  
 }  
}

Листинг 3.5.2 - Тест класса LogEntryParser

Этому подходу следуют многие классы .NET Framework, например, XmlReader/XmlWriter, XElement/XDocument и другие. TextReader/TextWriter представляют собой абстракции для работы с текстовыми данными, которые устраняют необходимость самописных интерфейсов, таких как IReader/IWriter.