

Лабораторная работа №10

Работа с файловым потоком

Цель работы: получить навыки работы с потоковым чтением и записью на языке C#.

Теоретическая часть

При работе с файлом можно создать один единый поток, в котором будет производиться чтение из файла и запись в файл.

Для создания потока можно использовать класс **FileStream**. У этого класса есть множество конструкторов, которые позволяют открыть файл в различных режимах.

```
public class FileStream : Stream
{
    ... public FileStream(SafeFileHandle handle, FileAccess access);
    ... public FileStream(IntPtr handle, FileAccess access);
    ... public FileStream(string path, FileMode mode);
    ... public FileStream(SafeFileHandle handle, FileAccess access, int bufferSize);
    ... public FileStream(IntPtr handle, FileAccess access, bool ownsHandle);
    ... public FileStream(string path, FileMode mode, FileAccess access);
    ... public FileStream(SafeFileHandle handle, FileAccess access, int bufferSize, bool isAsync);
    ... public FileStream(IntPtr handle, FileAccess access, bool ownsHandle, int bufferSize);
    ... public FileStream(string path, FileMode mode, FileAccess access, FileShare share);
    ... public FileStream(string path, FileMode mode, FileAccess access, FileShare share, int bufferSize);
    ... public FileStream(IntPtr handle, FileAccess access, bool ownsHandle, int bufferSize, bool isAsync);
    ... public FileStream(string path, FileMode mode, FileAccess access, FileShare share, int bufferSize, bool useAsync);
    ... public FileStream(string path, FileMode mode, FileAccess access, FileShare share, int bufferSize, FileOptions options);
}
```

Рисунок 1 – Конструкторы класса FileStream

При создании потока можно указать режим работы с файлом, в котором будет создаваться поток. Для этого используется перечисление **FileMode**.

```
public enum FileMode
{
    ... CreateNew = 1,
    ... Create = 2,
    ... Open = 3,
    ... OpenOrCreate = 4,
    ... Truncate = 5,
    ... Append = 6
}
```

Рисунок 2 – Перечисление FileMode

Также можно указать режим доступа к файлу из этого потока с помощью перечисления **FileAccess**.

```
public enum FileAccess
{
    ... Read = 1,
    ... Write = 2,
    ... ReadWrite = 3
}
```

Рисунок 3 – Перечисление FileAccess

Попробуем создать новый поток для чтения и записи.

```
FileStream stream = new FileStream(
    ... @"D:\User\Source\NewDir\file.txt",
    ... FileMode.OpenOrCreate,
    ... FileAccess.ReadWrite,
    ... FileShare.Read);
```

Рисунок 4 – Создание потока

В классах **StreamReader** и **StreamWriter** есть множество конструкторов, которые принимают поток (класс **Stream**), по желанию можно указать нужную кодировку.

```
public StreamReader(Stream stream);
public StreamReader(string path);
public StreamReader(Stream stream, bool detectEncodingFromByteOrderMarks);
public StreamReader(Stream stream, Encoding encoding);
public StreamReader(string path, bool detectEncodingFromByteOrderMarks);
public StreamReader(string path, Encoding encoding);
public StreamReader(Stream stream, Encoding encoding, bool detectEncodingFromByteOrderMarks);
public StreamReader(string path, Encoding encoding, bool detectEncodingFromByteOrderMarks);
public StreamReader(Stream stream, Encoding encoding, bool detectEncodingFromByteOrderMarks, int bufferSize);
public StreamReader(string path, Encoding encoding, bool detectEncodingFromByteOrderMarks, int bufferSize);
public StreamReader(Stream stream, Encoding? encoding = null, bool detectEncodingFromByteOrderMarks = true, int bufferSize = -1, bool leaveOpen = false);
```

Рисунок 5 – Конструкторы класса StreamReader

```
public StreamWriter(Stream stream);
public StreamWriter(string path);
public StreamWriter(Stream stream, Encoding encoding);
public StreamWriter(string path, bool append);
public StreamWriter(Stream stream, Encoding encoding, int bufferSize);
public StreamWriter(string path, bool append, Encoding encoding);
public StreamWriter(Stream stream, Encoding? encoding = null, int bufferSize = -1, bool leaveOpen = false);
public StreamWriter(string path, bool append, Encoding encoding, int bufferSize);
```

Рисунок 6 – Конструкторы класса StreamWriter

Теперь создадим объекты **reader** и **writer** для работы с файлом.

```
StreamReader reader = new StreamReader(stream);  
StreamWriter writer = new StreamWriter(stream);
```

Рисунок 7 – Создание объектов для работы с файлом

Не забываем сохранить изменения и очистить поток по окончании работы с файлом.

```
stream.Flush();  
stream.Dispose();
```

Рисунок 8 – Конец работы с файлом

Практическая часть

Разработать класс для работы с текстовым файлом. Класс должен выполнять следующие действия:

- 1) Создать новый файл (путь задается по умолчанию).
- 2) Создать новый файл по заданному пути.
- 3) Открыть существующий файл.
- 4) Прочитать строку из файла.
- 5) Прочитать count символов из файла.
- 6) Записать строку в файл.
- 7) Записать **count** символов в файл.
- 8) Для чтения и записи использовать классы **StreamReader** и **StreamWriter**.
- 9) Эти классы должны работать с одним и тем же потоком **FileStream**.

10) Конструкцию **using** использовать не нужно. Разработайте метод **Close** для сохранения изменений и освобождения ресурсов.

Содержание отчета

1. Титульный лист
2. Цель работы
3. Задание
4. Код программы
5. Результат выполнения