#13 Файловая система

Задание:

1. Создайте класс **FileSystemManager**. Реализуйте в нем следующую функциональность:
    a. Вывод информации о файле (параметр – *путь к файлу*):
        i. Дата создания;
        ii. Дата изменения;
        iii. Имя файла (с расширением).
    b. Поиск файла (параметры – *начальная директория* и *имя файла*). Метод должен произвести поиск файла внутри начальной директории, включая все подпапки и так рекурсивно.
    c. Поиск файла (параметры – *начальная директория* и *маска имени файла*). Работа метода схожа с методом выше за исключением того, что этот метод ищет файл по шаблону, используя регулярные выражения.
    d. Два предыдущих метода не должны дублировать код, переиспользуйте код, если это возможно.
    e. Вывод информации о дисковой системе:
        i. Свободное место на каждом томе;
        ii. Метку тома;
        iii. Полный объем тома.
    f. Создание папки по указанному пути.
    g. Создание файла с указанным содержимым (параметры – *имя файла*, *путь к файлу*, *содержимое файла в виде строки*, *режим создания файла*:
        i. **Create** – создает новый файл. Если таковой уже существует, то файл перезаписывается;
        ii. **Append** – открывает существующий файл и дозаписывает в него содержимое. Если файл не существует, то система создает новый.
        iii. **Exists** – выкидывает ошибку, если файл существует. В противном случае создает файл с содержимым).
    h. Считывает файл по указанному пути и возвращает считанный поток. Если такой файл не существует, приложение должно выбросить соответствующее исключение.
2. Обязательный отлов возможных ошибок;
3. В случае с потоками необходимо использовать конструкцию **using**.

4. Если необходимо «*построить*» путь, то следует использовать методы класса **Path**, но не использовать что-то типа:
   filePath+filename+ext;

Полезная информация и все классы, которые вам пригодятся для работы находятся в данном пространстве имен:

https://docs.microsoft.com/en-us/dotnet/api/system.io?view=netframework-4.8

Повышенный уровень:

1. Ознакомьтесь со способами обеспечения синхронизации потоков в C#:
   a. **Mutex**
   b. **lock**
   c. **Semaphore**
   d. **Timer**
   e. **Monitor**
   f. **AutoResetEvent**
2. Подробней про них можно прочесть на MSDN, классы расположены в пространстве имен **System.Threading**:
   https://docs.microsoft.com/en-us/dotnet/api/system.threading?view=netframework-4.8
   и на *metanit*:
   https://metanit.com/sharp/tutorial/11.4.php
3. Используя код из примера, создайте метод **ThreadExecution**, который в цикле создает 5 потоков, которые считывают один файл.

```
for (int i = 0; i < 5; i++)
{
    var thread = new Thread(start: () =>
    {
        using (var file = File.Open(path: @"E:\Work\data.txt", FileMode.Open))
        using (var streamReader = new StreamReader(file))
        {
            Console.WriteLine(streamReader.ReadToEnd());
        }
    });

    thread.Start();
}
```

4. Запустите приложение и удостоверьтесь, что система выбрасывает исключение «Файл занят другим процессом».
5. Измените код, добавив синхронизацию потоков, двумя способами:
    a. Используя **lock**;
    b. Используя любой другой метод (семафоры, мониторы, мьютексы и т.д.)
6. Продемонстрируйте работу программы без ошибки доступа к файлу.

Вопросы:

1. Охарактеризуйте классы, которые вы использовали в работе.
2. Для чего служит класс **FileInfo**?
3. Какой метод служит для проверки существования файла? Директории?
4. Как работает конструкция **using**?
5. Что такое **Stream**? Что такое **StreamWriter** и **StreamReader**?
6. Для чего служит класс **Path**?
7. Что такое «регулярные выражения»? Какое пространство имен в .NET отвечает за них?
8. Что такое **BinaryWriter** и **BinaryReader**?
9. Как можно записать данные в файл?

Повышенный уровень:

1. Охарактеризуйте возможности синхронизации потоков в .NET.
2. Для чего необходима синхронизация потоков?
3. Как работают следующие механизмы:
    a. **Monitor**
    b. **Semaphore**
    c. **Mutex**
    d. **lock**
    e. **AutoResetEvent**
    f. **Timer**