

INTERNET ACADEMY

Institute of Web Design & Software Services

C#7

インターネット・アカデミー

C# 目次

1. 日時の操作

2. ファイル操作



1.日時の操作

現在時刻の取得と表示

DateTime構造体 … 時刻の処理を行う**構造体**

クラスに似たもの

▼記述例

```
DateTime t = DateTime.Now;  
Console.WriteLine(t);
```

▼実行結果

```
5/21/2022 10:50:28 AM
```

現在時刻の取得と表示

任意の形式に変換した時刻の表示が可能

▼記述例

```
DateTime t = DateTime.Now;  
string result = t.ToString("yyyy/MM/dd hh:mm:ss");  
Console.WriteLine(result);
```

yyyy (年)、MM (月)、dd (日)
hh (時間)、mm (分)、ss(秒)

▼実行結果

2022年05月21日 10時54分03秒

現在時刻の取得と表示

任意の形式に変換した時刻の表示が可能

▼記述例

```
DateTime t = DateTime.Now;  
string result = t.ToString("yyyy年MM月dd日 hh時mm分ss秒");  
Console.WriteLine(result);
```

▼実行結果

2022/05/21 10:52:56

任意の時刻を表示

DateTimeのコンストラクタの引数で任意の時刻を設定可能

▼記述例

```
DateTime t = new DateTime(2022, 10, 1, 12, 34, 56);  
Console.WriteLine(t);
```

コンストラクタの引数として設定

▼実行結果

10/1/2022 12:34:56 PM

任意の時刻を表示

DateTimeのParseメソッドで任意の時刻を設定可能

▼記述例

```
string st = "2022/10/01 12:34:56";  
DateTime t = DateTime.Parse(st);  
Console.WriteLine(t);
```

▼実行結果

```
10/1/2022 12:34:56 PM
```


時間・分・秒の差分を求める

Timespan構造体を用いてDateTime構造体の差分を求めることができる

▼記述例

```
DateTime oldDateTime = new DateTime(2010, 10, 01, 12, 34, 56);  
TimeSpan ts = DateTime.Now - oldDateTime;  
Console.WriteLine(ts);
```

DateTimeの差分で時間の差分を計算

▼実行結果

4249.22:42:49.9414190

日・時・分・秒…の順で表示

時間・分・秒の差分を求める

Timespanの差分は日・時・分・秒…の単位でも取得可能

▼記述例

```
DateTime oldDateTime = new DateTime(2010, 10, 01, 12, 34, 56);  
TimeSpan ts = DateTime.Now - oldDateTime;  
Console.WriteLine(ts.Days);           // 日           → 4249  
Console.WriteLine(ts.Hours);          // 時           → 22  
Console.WriteLine(ts.Minutes);        // 分           → 42  
Console.WriteLine(ts.Seconds);        // 秒           → 49
```

日の差分を求める

Timespan構造体を用いてDateTime構造体の差分を求めることができる

▼記述例

```
DateTime oldDateTime = new DateTime(2010, 10, 01, 12, 34, 56);  
TimeSpan ts = DateTime.Now - oldDateTime;  
Console.WriteLine(ts.Days + "日の差があります。");
```

▼実行結果

4249日の差があります。



2. ファイル操作

ファイル関連のクラス

ファイル関連の操作を行うクラスには「**using System.IO;**」が必要

主なクラス

- ・ **StreamWriter** ... ファイルの書き込みクラス
- ・ **StreamReader** ... ファイルの読み込みクラス
- ・ **File** ... ファイルを作成、コピー、削除、移動するためのクラス
- ・ **FileStream** ... 様々なファイル操作（ファイルの作成に利用）
- ・ **FileInfo** ... ファイルを作成、コピー、削除、移動するためのクラス

ファイルの書き込み(新規)

C#でファイルの書き込みを行うには**StreamWriter**クラスを使う

▼記述例

```
string name = @"C:¥¥MyFolder¥¥sample.txt";  
using (StreamWriter sw = new StreamWriter(name, false, Encoding.UTF8))  
{  
    sw.WriteLine("Apple"); // 改行あり  
    sw.Write("Banana");    // 改行無し  
    sw.WriteLine("Orange"); // 改行あり  
}
```

ファイルの書き込み(新規)

C#でファイルの書き込みを行うには**StreamWriter**クラスを使う

▼記述例

```
string name = @"C:¥¥MyFolder¥¥sample.txt";
using (StreamWriter sw = new StreamWriter(name, false, Encoding.UTF8))
{
    sw.WriteLine("Apple"); // 改行あり
    sw.Write("Banana");    // 改行無し
    sw.WriteLine("Orange"); // 改行あり
}
```

ファイル名 (@から始める)

ファイルの書き込み(新規)

C#でファイルの書き込みを行うには**StreamWriter**クラスを使う

▼記述例

```
string name = @"C:¥¥MyFolder¥¥sample.txt";  
using (StreamWriter sw = new StreamWriter(name, false, Encoding.UTF8))  
{  
    sw.WriteLine("Apple"); // 改行あり  
    sw.Write("Banana");    // 改行無し  
    sw.WriteLine("Orange"); // 改行あり  
}
```

ファイル名 (@から始める)

！ポイント

「¥」はエスケープシーケンスを用いて「¥¥」と表現する

ファイルの書き込み(新規)

C#でファイルの書き込みを行うには**StreamWriter**クラスを使う

▼記述例

文字コード : UTF8

```
string name = @"C:¥¥MyFolder¥¥sample.txt";  
using (StreamWriter sw = new StreamWriter(name, False, Encoding.UTF8))  
{  
    sw.WriteLine("Apple"); // 改行あり  
    sw.Write("Banana");    // 改行無し  
    sw.WriteLine("Orange"); // 改行あり  
}
```

false:新規作成
true:既存のファイルに追加

ファイルの書き込み(新規)

C#でファイルの書き込みを行うには**StreamWriter**クラスを使う

▼記述例

```
string name = @"C:¥¥MyFolder¥¥sample.txt";  
using (StreamWriter sw = new StreamWriter(name, false, Encoding.UTF8))  
{  
    sw.WriteLine("Apple"); // 改行あり  
    sw.Write("Banana");    // 改行無し  
    sw.WriteLine("Orange"); // 改行あり  
}
```

ファイルの書き込み (Console.Write／Console.WriteLineと同じ考え方)

ファイルの書き込み(新規)

C#でファイルの書き込みを行うには**StreamWriter**クラスを使う

▼記述例

```
string name = @"C:¥¥MyFolder¥¥sample.txt";  
using (StreamWriter sw = new StreamWriter(name, false, Encoding.UTF8))  
{  
    sw.WriteLine(リソース（書き込みが終わったら解放が必要）)  
    sw.Write("Banana");    // 改行無し  
    sw.WriteLine("Orange"); // 改行あり  
}
```

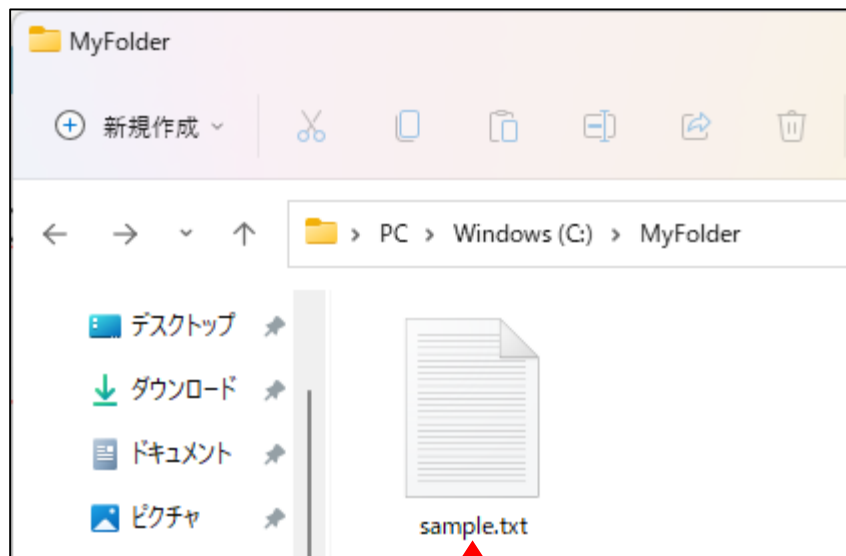
!

usingディレクティブを使うと、()内のリソースを使いきると自動解放してくれる

ファイルの書き込み(新規)

「MyFolder」フォルダ内に「sample.txt」というファイルが出来上がっている

▼MyFolderの中身



完成したファイル

▼「sample.txt」の中身

Apple
BananaOrange

ファイルの書き込み(追加)

ファイルを追記するにはStreamWriterクラスで引数のboolをtrueにする

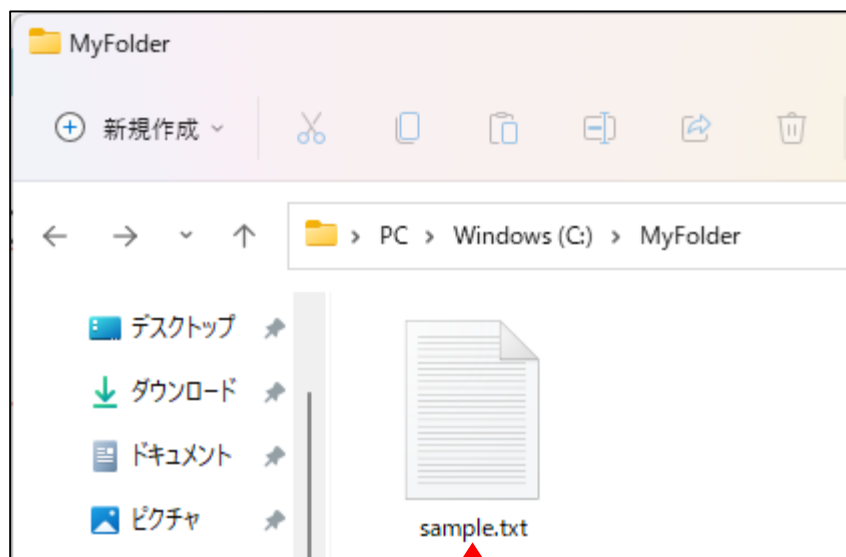
▼記述例

```
string name = @"C:¥¥MyFolder¥¥sample.txt";  
using (StreamWriter sw = new StreamWriter(name, true, Encoding.UTF8))  
{  
    sw.WriteLine("Apple"); // 改行あり  
    sw.Write("Banana");    // 改行無し  
    sw.WriteLine("Orange"); // 改行あり  
}
```

ファイルの書き込み(追加)

「MyFolder」フォルダ内に「sample.txt」というファイルが出来上がっている

▼MyFolderの中身



もともとあるファイル

▼「sample.txt」の中身

Apple
BananaOrange
Apple
BananaOrange

追加された部分

ファイルの書き込み

//を書くと、その後が実行されない**文（コメント）**を書くことができる

▼記述例

```
// コンソールにHelloWorldと表示して終了  
Console.WriteLine("Hello, World!");
```

1行のみのコメント=行コメント

▼実行結果

```
Hello, World!
```

コメントに記述した分は、実行結果に出力されない

ファイルの読み込み（ファイル全部読み込み）

C#でファイルの書き込みを行うには**StreamReader**クラスを使う

▼記述例

```
using (StreamReader sr = new StreamReader("C:¥¥MyFolder¥¥sample.txt"))  
{  
    string text = sr.ReadToEnd();  
    Console.WriteLine(text);  
}
```

▼実行結果

Apple

BananaOrange

ファイルの読み込み（1行ずつ読み込み）

C#でファイルの書き込みを行うには**StreamReader**クラスを使う




▼記述例

```
StreamReader sr = new StreamReader(@"C:¥¥MyFolder¥¥sample.txt");  
while (sr.EndOfStream == false)  
{  
    Console.WriteLine(sr.ReadLine());  
}  
sr.Close();
```

ファイルの読み込み（1行ずつ読み込み）

C#でファイルの書き込みを行うには**StreamReader**クラスを使う

▼記述例

```
StreamReader sr = new StreamReader(@"C:¥¥MyFolder¥¥sample.txt");  
while (sr.EndOfStream == false)  読み込み終了の判定  
{  
    Console.WriteLine(sr.ReadLine());  一行読み込み  
}  
sr.Close();  ファイルのクローズ（読み取り終了の宣言）
```

ファイルの存在の確認

C#でファイルの存在を確認するにはFileクラスの静的メソッドExistsを利用

▼記述例

```
string path1 = @"C:¥¥MyFolder¥¥sample.txt"; // 存在するファイル
string path2 = @"C:¥¥MyFolder¥¥dummy.txt"; // 存在しないファイル
Console.WriteLine(File.Exists(path1));
Console.WriteLine(File.Exists(path2));
```

▼実行結果

True → **存在する**
False → **存在しない**

ファイルの存在の確認

C#でファイルの存在を確認するにはFileクラスの静的メソッドExistsを利用

▼記述例

```
string path1 = @"C:¥¥MyFolder¥¥sample.txt"; // 存在するファイル
string path2 = @"C:¥¥MyFolder¥¥dummy.txt"; // 存在しないファイル
Console.WriteLine(File.Exists(path1));
Console.WriteLine(File.Exists(path2));
```

▼実行結果

True → 存在する

False → 存在しない

true・falseは表示した際にTrue・Falseになる

新規ファイルの作成

C#でファイルの作成には**FileStream**、**FileInfo**クラスを利用する

▼記述例

```
FileStream fs = File.Create(@"C:¥¥MyFolder¥¥sample2.txt");  
fs.Close();
```

▼記述例

```
FileInfo fi = new FileInfo(@"C:¥¥MyFolder¥¥sample3.txt");  
FileStream fs = fi.Create();  
fs.Close();
```


演習問題 1 : アクセスカウンターの作成

以下の仕様をもつアクセスカウンタークラスを定義しなさい。

- ・ クラス名はAccessCounter
- ・ ファイルパスを表すstring型のprivateなフィールドpathを持つ
- ・ 引数付きコンストラクタを持ちそこにstring型引数pathを与える
- ・ 引数付きコンストラクタでは引数をフィールドpathに代入する
- ・ publicなメソッドLogでpathで指定した箇所にアクセスログを残す
- ・ Logは戻り値なしで引数としてアクセス数（int）を与える
- ・ ログは「2022年05月28日 09時30分21秒:100アクセス」の様に記録
- ・ ログは上書き可能にすること

演習問題 1 : アクセスカウンターの作成

▼正解コード

```
public class AccessCounter
{
    private string path = "";
    public AccessCounter(string path)
    {
        this.path = path;
    }
    public void Log(int access)
    {
        using (StreamWriter sw = new StreamWriter(this.path, true, Encoding.UTF8))
        {
            DateTime t = DateTime.Now;
            string result = t.ToString("yyyy年MM月dd日 hh時mm分ss秒:");
            sw.Write(result);
            sw.WriteLine("{0}アクセス", access);
        }
    }
}
```


演習問題 1 : アクセスカウンターの作成

▼正解コード


```
public class AccessCounter
{
    private string path = "";
    public AccessCounter(string path)
    {
        this.path = path;
    }
    public void Log(int access)
    {
        using (StreamWriter sw = new StreamWriter(this.path, true, Encoding.UTF8))
        {
            DateTime t = DateTime.Now;
            string result = t.ToString("yyyy年MM月dd日 hh時mm分ss秒:");
            sw.Write(result);
            sw.WriteLine("{0}アクセス", access);
        }
    }
}
```

ログのファイルパス表すフィールド

演習問題 1 : アクセスカウンターの作成

▼正解コード

```
public class AccessCounter
{
    private string path = "";
    public AccessCounter(string path)
    {
        this.path = path;
    }
    public void Log(int access)
    {
        using (StreamWriter sw = new StreamWriter(this.path, true, Encoding.UTF8))
        {
            DateTime t = DateTime.Now;
            string result = t.ToString("yyyy年MM月dd日 hh時mm分ss秒:");
            sw.Write(result);
            sw.WriteLine("{0}アクセス", access);
        }
    }
}
```



引数付きのコンストラクタ

演習問題 1 : アクセスカウンターの作成

▼正解コード

```
public class AccessCounter
{
    private string path = "";
    public AccessCounter(string path)
    {
        this.path = path;
    }
    public void Log(int access)
    {
        using (StreamWriter sw = new StreamWriter(this.path, false, Encoding.UTF8))
        {
            DateTime t = DateTime.Now;
            string result = t.ToString("yyyy年MM月dd日 hh時mm分ss秒:");
            sw.Write(result);
            sw.WriteLine("{0}アクセス", access);
        }
    }
}
```

ログへの記録

演習問題 1 : アクセスカウンターの作成

▼正解コード

```
public class AccessCounter
{
    private string path = "";
    public AccessCounter(string path)
    {
        this.path = path;
    }
    public void Log(int access)
    {
        using (StreamWriter sw = new StreamWriter(this.path, true, Encoding.UTF8))
        {
            DateTime t = DateTime.Now;
            string result = t.ToString("yyyy年MM月dd日 hh時mm分ss秒:");
            sw.Write(result);
            sw.WriteLine("{0}アクセス", access);
        }
    }
}
```

上書き可能

true

演習問題 2 : アクセスカウンターの作成

演習問題 1 のクラスのインスタンスを生成しアクセスログを生成しなさい

- ・ アクセスログのパスは「C:¥MyFolder¥accesslog.txt」 とすること
- ・ ファイル実行後ログの中身を確認しなさい

▼実行結果（accesslog.txt）の中身

2022年05月28日 09時48分46秒:100アクセス

演習問題 2 : アクセスカウンターの作成

string型の変数でパス名を定義しインスタンスを生成しLogメソッドを実行します

- ・ Logの引数を100として実行

※メソッドを実行した現在時刻とアクセス数が正しく記録されていれば完成

▼正解コード

```
string path = @"C:¥¥MyFolder¥¥accesslog.txt";  
AccessCounter ac = new AccessCounter(path);  
ac.Log(100);
```

演習問題 3 : アクセスカウンターの作成

演習問題 2 のプログラムを再び実行しログが上書きされていることを確認しなさい

▼実行結果（accesslog.txt）の中身

2022年05月28日 09時48分46秒:100アクセス

2022年05月28日 09時48分55秒:100アクセス

演習問題 2 : アクセスカウンターの作成

実行しファイルが書き込まれていることが確認できていれば正解です

※ログが一つしかない場合にはStreamWriterクラスのコンストラクタに問題がある可能性があるので確認してみましょう