講義メモ
・p.171「デストラクタ」から再開します

アレンジ演習フォロー：p.169 construct02.cs

・名前と年齢を引数で受け取るコンストラクタ④を追加しよう
・Mainメソッドにおいて、コンストラクタ④を呼ぶ処理と、結果を表示する処理を追加しよう

作成例

```
//アレンジ演習：p.169 construct02.cs
using System;
class MyClass {
    private string name;    //名前：外部から見えないインスタンス変数
    private int age;        //年齢：外部から見えないインスタンス変数
    private string address; //住所：外部から見えないインスタンス変数
    public void Show() { //戻り値のないインスタンスメソッド
        string toshi; //メソッド内で用いるローカル変数
        if (age == -1) { //年齢が-1になっていたら
            toshi = "不明";
        } else {
            toshi = age.ToString(); //int型から文字列化して代入
        }
        Console.WriteLine("氏名:{0} 住所:{1} 年齢:{2}", name, address, toshi);
    }
    public MyClass(string str) { //引数のあるコンストラクタ①(文字列：名前)
        name = str;
        address = "不定"; //住所は規定値
        age = -1; //年齢は規定値
    }
    public MyClass(int x) { //引数のあるコンストラクタ②(整数：年齢)
        age = x;
        name = "不明"; //名前は規定値
        address = "不定"; //住所は規定値
    }
    public MyClass(string str1, string str2, int x) { //引数のあるコンストラクタ③(文字
列：名前,文字列：住所,整数：年齢)
        name = str1;
        address = str2;
        age = x;
    }
    public MyClass(string str1, int x) { //【以下追加】引数のあるコンストラクタ④(文字列：
名前,整数：年齢)
        name = str1;
        age = x;
        address = "不定"; //住所は規定値
    }
}
class construct01 {
    public static void Main() {
        MyClass mc1 = new MyClass(18); //コンストラクタ②が呼ばれる
        MyClass mc2 = new MyClass("粂井康孝"); //コンストラクタ①が呼ばれる
        MyClass mc3 = new MyClass("田中太郎", "東京都", 32); //コンストラクタ③が呼ばれ
る
```

```
        MyClass mc4 = new MyClass("田中太郎", 32); //【追加】コンストラクタ④が呼ばれる
        mc1.Show();
        mc2.Show();
        mc3.Show();
        mc4.Show(); //【追加】
    }
}
```

p.171（コンストラクタのオーバーロードとデフォルトコンストラクタ）

・p.167の通り、コンストラクタを一切記述しないと、引数無し中身無しのコンストラクタが内部的に用意されて用いられる
・これに対して、p.169 construct02.csのように、引数有りのコンストラクタを記述できるが、こうすると、引数無し中身無しのコンストラクタは用意されなくなる
・よって、必要があれば、自前で、引数無し中身無しのコンストラクタを記述して、オーバーロードにすること

アレンジ演習：p.169 construct02.cs

・名前を"不明"、住所を"不定"、年齢を-1にする、引数で受け取らないコンストラクタ⑤を追加しよう
・Mainメソッドにおいて、コンストラクタ⑤を呼ぶ処理と、結果を表示する処理を追加しよう

作成例

```
//アレンジ演習：p.169 construct02.cs
using System;
class MyClass {
    private string name;    //名前：外部から見えないインスタンス変数
    private int age;        //年齢：外部から見えないインスタンス変数
    private string address; //住所：外部から見えないインスタンス変数
    public void Show() { //戻り値のないインスタンスメソッド
        string toshi; //メソッド内で用いるローカル変数
        if (age == -1) { //年齢が-1になっていたら
            toshi = "不明";
        } else {
            toshi = age.ToString(); //int型から文字列化して代入
        }
        Console.WriteLine("氏名:{0} 住所:{1} 年齢:{2}", name, address, toshi);
    }
    public MyClass(string str) { //引数のあるコンストラクタ①(文字列：名前)
        name = str;
        address = "不定"; //住所は規定値
        age = -1; //年齢は規定値
    }
    public MyClass(int x) { //引数のあるコンストラクタ②(整数：年齢)
        age = x;
        name = "不明"; //名前は規定値
        address = "不定"; //住所は規定値
    }
    public MyClass(string str1, string str2, int x) { //引数のあるコンストラクタ③(文字列：名前,文字列：住所,整数：年齢)
        name = str1;
        address = str2;
        age = x;
    }
```

```
    public MyClass(string str1, int x) { //引数のあるコンストラクタ④(文字列:名前,整数:
年齢)
        name = str1;
        age = x;
        address = "不定"; //住所は規定値
    }
    public MyClass() { //【以下追加】引数の無いコンストラクタ⑤
        name = "不明"; //名前は規定値
        age = -1; //年齢は規定値
        address = "不定"; //住所は規定値
    }
}
class construct01 {
    public static void Main() {
        MyClass mc1 = new MyClass(18); //コンストラクタ②が呼ばれる
        MyClass mc2 = new MyClass("粂井康孝"); //コンストラクタ①が呼ばれる
        MyClass mc3 = new MyClass("田中太郎", "東京都", 32); //コンストラクタ③が呼ばれ
る
        MyClass mc4 = new MyClass("田中太郎", 32); //コンストラクタ④が呼ばれる
        MyClass mc5 = new MyClass(); //【追加】コンストラクタ⑤が呼ばれる
        mc1.Show();
        mc2.Show();
        mc3.Show();
        mc4.Show();
        mc5.Show(); //【追加】
    }
}
```

p.171 デストラクタ

・プログラムの終了や有効期間の終了によってオブジェクトが破棄される時、その直前に自動的に呼び出されるのがデストラクタ
・よって、コンストラクタに似ているが、仕様の違いに注意
・主に、利用していたリソース(資源＝ファイルやデータベース、通信など)の解放のような「後始末」に向く。
・デストラクタはメソッドに似ているが、アクセス修飾子や引数、戻り値はない。
・引数がないのでオーナーライドもできない
・デストラクタは名前もなく「~クラス名」固定とする。
・書式: ~クラス名() { 内容 }
・コンストラクタと同様で、記述がなければ自動的に中身の無いデストラクタが内部的に用意されて用いられる
・デストラクタはインスタンスに含まれるので、複数のインスタンスを生成すると、複数のデストラクタが動作する
・この時の実行順序は決まっていない

p.172 destruct01.cs

```
//p.172 destruct01.cs
using System;
class DestructTest {
    int x;
    // デストラクタ
    ~DestructTest() {
        Console.WriteLine("デストラクタが呼ばれました");
        Console.WriteLine("xは{0}です", x);
    }
    // 引数付きコンストラクタ
```

```
    public DestructTest(int n) {
        Console.WriteLine("コンストラクタが呼ばれました");
        x = n;
        Console.WriteLine("xに{0}を代入しました", n);
    }
}
class destruct {
    public static void Main(){
        DestructTest dt1 = new DestructTest(1);
        Console.WriteLine("dt1生成");
        DestructTest dt2 = new DestructTest(2);
        Console.WriteLine("dt2生成");
        DestructTest dt3 = new DestructTest(3);
        Console.WriteLine("dt3生成"); //この直後に3つのデストラクタが順不同で実行される
    }
}
```

アレンジ演習：p.172 destruct01.cs

・オブジェクトが参照変数からアクセスできなくなると、消去の対象となり、消去時にデストラクタが動作する
・しかし、メモリや処理の余裕がある間は、この作業（ガベージコレクション）は実行されない
・このことを試すために、DestructTest型の参照変数dt1を再利用してみよう

```
        DestructTest dt1 = new DestructTest(1); //オブジェクト①生成
        dt1 = new DestructTest(2); //オブジェクト②生成、ここで①が消去対象になる
        dt1 = new DestructTest(3); //オブジェクト③生成、ここで②も消去対象になる
```

作成例

```
//アレンジ演習：p.172 destruct01.cs
using System;
class DestructTest {
    int x;
    // デストラクタ
    ~DestructTest() {
        Console.WriteLine("デストラクタが呼ばれました");
        Console.WriteLine("xは{0}です", x);
    }
    // 引数付きコンストラクタ
    public DestructTest(int n) {
        Console.WriteLine("コンストラクタが呼ばれました");
        x = n;
        Console.WriteLine("xに{0}を代入しました", n);
    }
}
class destruct {
    public static void Main(){
        DestructTest dt1 = new DestructTest(1); //オブジェクト①生成
        dt1 = new DestructTest(2); //オブジェクト②生成、ここで①が消去対象になる
        dt1 = new DestructTest(3); //オブジェクト③生成、ここで②も消去対象になる
    } //実際は①②もプログラム終了まで残ってしまい、まとめてデストラクタが動作する
}
```

p.174 this

・メソッドやコンストラクタの中で、自分が所属するオブジェクトへの参照を得たい場合に用いる
・よって、p.174上の例のように、インスタンス変数であることを「this.」を前置して明示することもできるが、通常は省略する
【補足①】
・thisの用途として、コンストラクタの引数名をデータメンバ名と同じにする手法が良く用いられる
・これで、インスタンス変数に「this.」を前置して「this.メンバ名 = メンバ名;」と表記できる
例:
```
class Slime {
  int hp, mp;
  public Slime(hp, mp) { this.hp = hp; this.mp = mp; } //コンストラクタの引数がメンバ名と同じ
}
```
【補足②】
・thisの用途として、コンストラクタのオーバロードにおいて、他のコンストラクタをthisで呼ぶ手法が良く用いられる
・書式: public クラス名(引数リスト) : this(引数リスト) {}
・この書式をコンストラクタ初期化子という
・これで、複数のコンストラクタにまたがって同一内容を記述することを避けられる
例:
```
class Slime {
  int hp, mp;
  public Slime(int hp, int mp) { this.hp = hp; this.mp = mp; } //HPとMPを指定するコンストラクタ①
  public Slime(int hp) : this(hp, 0) {} //HPだけを指定するコンストラクタ②は①を利用
  public Slime() : this(0, 0) {} //引数を指定しないコンストラクタ③も①を利用
}
```

アレンジ演習:p.169 construct02.cs ⇒ p.169 construct02a.cs

・【補足①】を用いて、3つのコンストラクタの引数名をメンバ名と同じにしよう

作成例

```
//アレンジ演習:p.169 construct02.cs
using System;
class MyClass {
    private string name;    //名前:外部から見えないインスタンス変数
    private int age;        //年齢:外部から見えないインスタンス変数
    private string address; //住所:外部から見えないインスタンス変数
    public void Show() { //戻り値のないインスタンスメソッド
        string toshi; //メソッド内で用いるローカル変数
        if (age == -1) { //年齢が-1になっていたら
            toshi = "不明";
        } else {
            toshi = age.ToString(); //int型から文字列化して代入
        }
        Console.WriteLine("氏名:{0} 住所:{1} 年齢:{2}", name, address, toshi);
    }
    public MyClass(string name) { //【変更】引数のあるコンストラクタ①(文字列:名前)
        this.name = name; //【変更】
        address = "不定"; //住所は規定値
        age = -1; //年齢は規定値
    }
    public MyClass(int age) { //【変更】引数のあるコンストラクタ②(整数:年齢)
```

```
        this.age = age; //【変更】
        name = "不明"; //名前は規定値
        address = "不定"; //住所は規定値
    }
    public MyClass(string name, string address, int age) { //引数のあるコンストラクタ③
(文字列:名前,文字列:住所,整数:年齢)
        this.name = name; //【変更】
        this.address = address; //【変更】
        this.age = age; //【変更】
    }
}
class construct01 {
    public static void Main() {
        MyClass mc1 = new MyClass(18); //コンストラクタ②が呼ばれる
        MyClass mc2 = new MyClass("粂井康孝"); //コンストラクタ①が呼ばれる
        MyClass mc3 = new MyClass("田中太郎", "東京都", 32); //コンストラクタ③が呼ばれ
る
        mc1.Show();
        mc2.Show();
        mc3.Show();
    }
}
```

アレンジ演習:p.169 construct02a.cs

・さらに【補足②】を用いて、コンストラクタ①と②はコンストラクタ③を用いるようにしよう

作成例

```
//アレンジ演習:p.169 construct02.cs
using System;
class MyClass {
    private string name;    //名前:外部から見えないインスタンス変数
    private int age;        //年齢:外部から見えないインスタンス変数
    private string address; //住所:外部から見えないインスタンス変数
    public void Show() { //戻り値のないインスタンスメソッド
        string toshi; //メソッド内で用いるローカル変数
        if (age == -1) { //年齢が-1になっていたら
            toshi = "不明";
        } else {
            toshi = age.ToString(); //int型から文字列化して代入
        }
        Console.WriteLine("氏名:{0} 住所:{1} 年齢:{2}", name, address, toshi);
    }
    public MyClass(string name) : this(name, "不定", -1) { } //【変更】コンストラクタ①(
文字列:名前)⇒③を呼ぶ
    public MyClass(int age) : this("不明", "不定", age) { }  //【変更】コンストラクタ②(
整数:年齢)⇒③を呼ぶ
    public MyClass(string name, string address, int age) { //引数のあるコンストラクタ③
(文字列:名前,文字列:住所,整数:年齢)
        this.name = name;
        this.address = address;
        this.age = age;
    }
```

```
}
class construct01 {
    public static void Main() {
        MyClass mc1 = new MyClass(18); //コンストラクタ②が呼ばれる
        MyClass mc2 = new MyClass("粂井康孝"); //コンストラクタ①が呼ばれる
        MyClass mc3 = new MyClass("田中太郎", "東京都", 32); //コンストラクタ③が呼ばれる
        mc1.Show();
        mc2.Show();
        mc3.Show();
    }
}
```

p.176 既存のクラスを使ってみる

・ここまでで、System、Mathなどの既存のクラスを利用しているが、ここでは、インスタンスメソッドを持つクラスの一例として、ArrayListクラスを用いよう
・ArrayListクラスは、配列の機能を拡張した仕組みを提供するもので、配列にはない柔軟な利用が可能
・利用には「using Sytem.Collections;」を指定する
　※ ArrayListクラスなどのデータ構造を表すことのできるクラスをコレクションと呼ぶ
・インスタンスを生成すると、配列に似たデータ構造が生成されるが、配列とは異なり、要素数を事前に決める必要がなく、動的に変更することもできる
・要素の追加にはインスタンスメソッドのAdd(データ)を用い、追加にしたがって要素数が拡張される
・件数を返すCountプロパティも利用可能
　※ プロパティは特殊なメソッドで8章で説明
・格納済の要素は、配列と同様にオブジェクト名[添字]で直接アクセスできる

p.177 arraylist01.cs

```
//p.177 arraylist01.cs
using System;
using System.Collections; //ArrayList用
class arraylist01 {
    public static void Main() {
        bool bEnd = false; //終了フラグをオフで初期化
        string strData; //読込用
        double sum = 0.0; //合計
        ArrayList al = new ArrayList(); //ArrayListオブジェクトを生成
        while (true) { //無限ループ
            Console.Write("データ(数値以外入力で終了)-- ");
            strData = Console.ReadLine();
            if (!Char.IsDigit(strData[0]) && strData[0] != '-') { //1文字目が数字ではなくマイナスでもない？
                bEnd = true; //終了フラグをオンにする
            } else { //数字またはマイナスなら
                al.Add(double.Parse(strData)); //実数に変換してArrayListに格納
            }
            if (bEnd) { //終了フラグがオン？
                break; //繰返しを抜ける
            }
        }
        for (int i = 0; i < al.Count; i++) { //ArrayListに格納した件数の分、繰返す
            Console.WriteLine("Data[{0}] = {1}", i + 1, al[i]); //値を表示
            sum += (double)al[i]; //合計に足し込む
```

```
        }
        int count = al.Count; //ArrayListに格納した件数を得る
        double avr = sum / count; //合計を件数で割って平均値を得る
        Console.WriteLine("データ個数 = {0}", count);
        Console.WriteLine("平均値 = {0}", avr);
    }
}
```

アレンジ演習:p.177 arraylist01.cs

・Countプロパティを2回呼び出しているが、1回にしよう
・終了フラグを廃止しよう

作成例

```
//アレンジ演習:p.177 arraylist01.cs
using System;
using System.Collections; //ArrayList用
class arraylist01 {
    public static void Main() {
        string strData; //読込用
        double sum = 0.0; //合計
        ArrayList al = new ArrayList(); //ArrayListオブジェクトを生成
        while (true) { //無限ループ
            Console.Write("データ(数値以外入力で終了)-- ");
            strData = Console.ReadLine();
            if (!Char.IsDigit(strData[0]) && strData[0] != '-') { //1文字目が数字
ではなくマイナスでもない？
                break; //【移動】繰返しを抜ける
            } else { //数字またはマイナスなら
                al.Add(double.Parse(strData)); //実数に変換してArrayListに格納
            }
        }
        int count = al.Count; //【移動】ArrayListに格納した件数を得る
        for (int i = 0; i < count; i++) { //【変更】件数の分、繰返す
            Console.WriteLine("Data[{0}] = {1}", i + 1, al[i]); //値を表示
            sum += (double)al[i]; //合計に足し込む
        }
        Console.WriteLine("データ個数 = {0}", count);
        Console.WriteLine("平均値 = {0}", sum / count); //【変更】合計を件数で割って平
均値を得る
    }
}
```

p.180 練習問題1 ヒント

・クラス名や変数名は自由
・例: class MyClass { public int i; }
・読みだした値は確認用に表示すると良い

作成例

```
//p.180 練習問題1
using System;
```

```
class MyClass { public int i; } //int型のpublicインスタンス変数のみを持つクラス
class ex0701 {
    public static void Main() {
        MyClass mc = new MyClass(); //MyClassオブジェクトを生成
        mc.i = 10;
        Console.WriteLine("mc.i = {0}", mc.i);
    }
}
```

p.180 練習問題2 ヒント

・クラス名やメソッド名は自由
・ただし、オーバーロードなので、同じメソッド名とすること
・Mainメソッドから呼び出すために、2つのメソッドはpublicにする
・int型とint型の和を求めるメソッドは戻り値型もintにする
・double型とdouble型の和を求めるメソッドは戻り値型もdoubleにする
・結果は確認用に表示すると良い

作成例

```
//p.180 練習問題2
using System;
class MyClass {
    public int sum(int a, int b) { //メソッド①
        return a + b;
    }
    public double sum(double a, double b) { //メソッド②＝①のオーバーロード
        return a + b;
    }
}
class ex0702 {
    public static void Main() {
        MyClass mc = new MyClass(); //MyClassオブジェクトを生成
        Console.WriteLine("整数 {0} + {1} = {2}", 2, 3, mc.sum(2, 3)); //メソッド①
が呼ばれる
        Console.WriteLine("実数 {0} + {1} = {2}", 3.14, 4.99, mc.sum(3.14,
4.99)); //メソッド②が呼ばれる
    }
}
```

第8章 クラスとメソッドの詳細

p.181 メソッドの再帰呼び出し

・最近のプログラム言語では、あるメソッドの中で自分自身を呼び出すことが可能で、これを再帰という。
・再帰を上手く利用すると、プログラムをシンプルに記述出来る場合がある
・単純に自分自身を呼び出すと呼び出しが無限ループするので（これ以上）自分を呼び出さずに式や値を返す処置が必要

p.181 階乗を計算する

・階乗とは、ある正の整数において、その数から1までの全整数の積
　※ 実質的にはその数から2までの全整数の積になる
・整数nの階乗を「n!」で表し、例えば、2!は2、3!は6、4!は24、5!は120、…

・これを逆順で展開すると
 5! = 5×4×3×2×1
 4! =   4×3×2×1
 3! =   3×2×1
・なので、5!=5×4!、4!=4×3! であることから「n! = n×(n-1)!」が導ける。
・これをメソッドFactにすると、
 int CalcFact(int n) { //nの階乗
  return n * CalcFact(n - 1); //n×(n-1)!を返す(再帰する)
 }
・これに、再帰の終了条件として「0の階乗は1」を加えると良い
 int CalcFact(int n) { //nの階乗
  return (n > 0) ? n * CalcFact(n - 1) : 1; //nが0超ならn×(n-1)!を返す(再帰する)でなければ1を返す
 }
・p.182 fact01.csは上記を展開したメソッドになっている

p.182 fact01.cs

```
//p.182 fact01.cs
using System;
class Fact {
    public long CalcFact(int n) { //n!を返す
        long fact; //階乗値
        if (n == 0) { //0!は1なので(再帰の終了条件になる)
            fact = 1; //1を返す
        } else { //1以上ならば
            fact = n * CalcFact(n - 1); //「n! = n * (n - 1)!」により再帰する
        }
        return fact; //階乗値を返す
    }
}
class fact01 {
    public static void Main() {
        Fact f = new Fact();
        for (int i = 0; i <= 20; i++) {
            Console.WriteLine("{0}! = {1}", i, f.CalcFact(i));
        }
    }
}
```

アレンジ演習:p.182 fact01.cs

・CalcFact(int n)メソッドを上記を用いてシンプルにしよう

提出:アレンジ演習:p.182 fact01.cs

次回予告:p.185「フィボナッチ数列を求める」