講義メモ
・p.147「暗黙の型指定がなされた配列」から

提出：アレンジ演習：p.146 jagged02.cs

・2つあるforを2重ループにして見やすくしよう
・ヒント：下記の●を0から1まで繰返せばよい

```
        for (int j = 0; j < name[●].Length; j++) { //含まれる配列の要素数について繰返す
            Console.WriteLine(name[●][j]); //含まれる配列の要素を表示
        }
```

作成例

```
//アレンジ演習：p.146 jagged02.cs
using System;
class jagged02 {
    public static void Main() {
        string[][] name = new string[2][]; //(2次元)ジャグ配列の宣言と上位次元の要素数＝含む配列の数は2と指定
        name[0] = new string[2]{"田中", "工藤"}; //含まれる配列①の初期化(2要素)
        name[1] = new string[3]{"吉田", "佐藤", "池田"}; //含まれる配列②の初期化(3要素)
        for (int i = 0; i < name.Length; i++) { //含まれる配列数(2)について繰返す
            for (int j = 0; j < name[i].Length; j++) { //含まれる配列のそれぞれの要素数(2,3)について繰返す
                Console.WriteLine(name[i][j]); //含まれる配列の要素を表示
            }
        }
    }
}
```

p.147 暗黙の型指定がなされた配列

・varキーワードにより、初期化された変数の型を自動決定できる(p.61)
・これは、配列に対しても適用可能
・「var []」や「var [][]」などになるわけではなく、配列としての型になる
・書式例： var 配列名 = new []{値,…};

p.147 var03.cs

```
//p.147 var03.cs
using System;
class var03
{
    public static int Main()
    {
        var name = new []{"太郎", "次郎", "三郎", "四郎"}; //nameはstring[]型となる
        for (var i = 0; i < name.Length; i++) { //Lengthは4になる
            Console.WriteLine(name[i]);
        }
        var f = new[] { 0.5, 0.9, 1.5, 2.3 }; //fはdouble型になる
        for (var i = 0; i < f.Length; i++) { //Lengthは4になる
            Console.WriteLine(f[i]);
```

```
        }
        Console.WriteLine("nameの型は{0}, naの型は{1}", name.GetType(),
f.GetType()); //System.String[],System.Double[]
        return 0;
    }
}
```

アレンジ演習：p.146 jagged02.cs ②

・配列nameをvarで暗黙の型指定ができるかどうか確認しよう
・可能であり、string[][]型になる。

作成例

```
//アレンジ演習：p.146 jagged02.cs
using System;
class jagged02 {
    public static void Main() {
        var name = new string[2][]; //(2次元)ジャグ配列の宣言と上位次元の要素数＝含む
配列の数は2と指定
        name[0] = new string[2]{"田中", "工藤"}; //含まれる配列①の初期化(2要素)
        name[1] = new string[3]{"吉田", "佐藤", "池田"}; //含まれる配列②の初期化(3
要素)
        for (int i = 0; i < name.Length; i++) { //含まれる配列数(2)について繰返す
            for (int j = 0; j < name[i].Length; j++) { //含まれる配列のそれぞれの要素
数(2,3)について繰返す
                Console.WriteLine(name[i][j]); //含まれる配列の要素を表示
            }
        }
    }
}
```

p.148 １次元配列のソート

・C#が提供するArrayというクラスがあり、配列を扱う便利なメソッド等を持っている
・その1つがArray.Sortで、引数として１次元配列名を指定すると、要素を昇順に整列する
・また、全要素を逆順にするArray.Reverseもあり、Array.Sortの後に実行することで降順の整列にできる

p.148 sort01.cs

```
//p.148 sort01.cs
using System;
class sort01
{
    public static void Main()
    {
        string[] name = new string[5]{"Eric", "Peter", "Frank", "Kate",
"Thomas"};
        for (int i = 0; i < name.Length; i++) { //全要素について
            Console.WriteLine(name[i]); //整列前を表示
        }
        Console.WriteLine(); //改行
        Array.Sort(name); //昇順に整列
        for (int i = 0; i < name.Length; i++) { //全要素について
```

```
            Console.WriteLine(name[i]); //整列後を表示
        }
        Console.WriteLine(); //改行
        Array.Reverse(name); //逆順にする(Sort後なので降順になる)
        for (int i = 0; i < name.Length; i++) { //全要素について
            Console.WriteLine(name[i]);
        }
    }
}
```

アレンジ演習 p.148 sort01.cs

・かなや漢字での動作を確認しよう
① new string[5]{"あ", "ぁ", "ア", "ァ", "亜"};
 ⇒ ァ、ぁ、ア、あ、亜 となる
② new string[5]{"一", "二", "三", "四", "五"};
 ⇒ 一、五、三、四、二 となる
③ new double[5]{3.14, -3.14, 0.0, -1 / 0.0, 1 / 0.0}; //「-1 / 0.0」は-∞、「1 / 0.0」は∞
 ⇒ -∞、-3.14、0、3.14、∞ となる

作成例③

```
//アレンジ演習：p.148 sort01.cs
using System;
class sort01
{
    public static void Main()
    {
        //string[] name = new string[5]{"あ", "ぁ", "ア", "ァ", "亜"}; //①
        //string[] name = new string[5]{"一", "二", "三", "四", "五"}; //②
        double[] name = new double[5]{3.14, -3.14, 0.0, -1 / 0.0, 1 / 0.0}; //③「-1 / 0.0」は-∞、「1 / 0.0」は∞
        for (int i = 0; i < name.Length; i++) { //全要素について
            Console.WriteLine(name[i]); //整列前を表示
        }
        Console.WriteLine(); //改行
        Array.Sort(name); //昇順に整列
        for (int i = 0; i < name.Length; i++) { //全要素について
            Console.WriteLine(name[i]); //整列後を表示 -∞、-3.14、0、3.14、∞ となる
        }
        Console.WriteLine(); //改行
        Array.Reverse(name); //逆順にする(Sort後なので降順になる)
        for (int i = 0; i < name.Length; i++) { //全要素について
            Console.WriteLine(name[i]);
        }
    }
}
```

p.150 foreach文による反復処理

・for文の変型で「データ構造(コレクション)に属する全データについて繰返す」のが、foreach文
・配列を指定すると「配列の全要素について繰返す」となるので便利
・foreachでは、作業用の変数を指定して、そこに1つずつデータ(要素値)が得られる

・書式: foreach(型 作業用の変数 in 配列等のコレクション名) { 作業用の変数を用いる処理 }
※ 作業用の変数の型は、配列等のコレクションの型にするが、varにすることが多い
※ 作業用の変数は要素値のコピーなので、値を書き換えても元の値には影響しない

p.150 foreach01.cs

```
//p.150 foreach01.cs
using System;
class foreach01
{
    public static void Main()
    {
        string[] Animal = new string[]{"犬", "猫", "雉", "猿"};
        int[] Num = new int[]{10, 20, 30, 40};
        foreach (string str in Animal) { //配列Animalの全要素について繰返す
            Console.WriteLine(str);
        }
        Console.WriteLine(); //改行
        foreach (int i in Num) { //配列Numの全要素について繰返す
            Console.WriteLine(i);
        }
    }
}
```

アレンジ演習 p.148 sort01.cs ②

・foreachを用いて簡略化しよう
・作業変数の型はvarにすること

```
//アレンジ演習:p.148 sort01.cs ②
using System;
class sort01
{
    public static void Main()
    {
        //string[] name = new string[5]{ァ、ぁ、ア、あ、亜};
        //string[] name = new string[5]{"一", "二", "三", "四", "五"};
        double[] name = new double[5]{3.14, -3.14, 0.0, -1 / 0.0, 1 / 0.0};
//「-1 / 0.0」は-∞、「1 / 0.0」は∞
        foreach (var w in name) { //【変更】全要素について
            Console.WriteLine(w); //整列前を表示
        }
        Console.WriteLine(); //改行
        Array.Sort(name); //昇順に整列
        foreach (var w in name) { //【変更】全要素について
            Console.WriteLine(w); //整列前を表示
        }
        Console.WriteLine(); //改行
        Array.Reverse(name); //逆順にする(Sort後なので降順になる)
        foreach (var w in name) { //【変更】全要素について
            Console.WriteLine(w); //整列前を表示
        }
    }
}
```

p.152 練習問題 ヒント

・p.136 Average02.csを元にすると良い
・実行イメージから考えよう

受験者数:3
1人目の点数:50
2人目の点数:60
3人目の点数:80
平均点は63.3
降順にすると
80
60
50

作成例

```
//p.152 練習問題
using System;
class average02 {
    public static void Main() {
        Console.Write("受験者数:"); int no = int.Parse(Console.ReadLine());
        int[] point = new int[no]; //int型の配列pointを受験者数の分生成
        int sum = 0; //合計
        for (int i = 0; i < no; i++) { //全要素について繰返す(※要素値を書き換えるので
foreachは不可)
            Console.Write("{0}番:", i + 1); point[i] =
int.Parse(Console.ReadLine());
            sum += point[i]; //要素値をsumに足し込む
        }
        double average = (double)sum / no; //合計を件数で割って平均値を得る
        Console.WriteLine("平均点 = {0:##.#}", average); //平均値(小数点以下1桁)を
表示
        Array.Sort(point); //昇順に整列
        Array.Reverse(point); //逆順にする(Sort後なので降順になる)
        Console.WriteLine("降順にすると");
        foreach (var w in point) { //全要素について繰返す
            Console.WriteLine(w);
        }
    }
}
```

第7章 クラスの基礎

p.153 クラスと何か

・C#などのオブジェクト指向言語では、プログラムをクラスを単位として作成し、データ構造もクラスを基本として扱うことができる
※ C++のように、プログラムの一部をクラスを単位として作成するものもある
・プログラムはビルドして実行を指示することにより、記述内容に従った実体がメモリ上に生成されて動作する
・つまり、プログラム＝クラスは設計図にあたり、動作する実体をオブジェクトという
・この考え方を、クラスの中で用いるデータ構造や部品に当てはめることもできる
・つまり、プログラムにおいて、他のクラスの実体を生成して用いることが、プログラムの部品化にあたる

・なお、クラスから生成したオブジェクトをインスタンスともいう

p.154 簡単なクラスを定義しよう

・最も簡単なクラスは変数のみを持つクラスで、複数の変数を持つクラスにより、C/C++でいう構造体に類似するデータ構造を表現できる
・書式: class クラス名 { アクセス修飾子 型 変数名; … }
・アクセス修飾子は主にpublicとprivateで、外部から直接アクセスを許す場合はpublicと明示する。内部用の場合は無指定にするか、privateと明示するじょうk
・例: class Monster { public string name; public int hp; public int mp; } //名前とHP,MPを持つ怪物
・クラスを型として変数を宣言することで、クラスから生成したインスタンスを扱うことができる
・生成には配列と同様にnew演算子を用いる
・宣言の書式: クラス名 インスタンス名;
・生成の書式: インスタンス名 = new クラス名();
・宣言と生成は同時に行うことが可能。書式: クラス名 インスタンス名 = new クラス名();
例: Monster veldra = new Monster(); //怪物クラスから実体としてveldraを生成
・複数のインスタンスをまとめて生成できる
例: Monster veldra = new Monster(), rimuru = new Monster(); //怪物クラスから実体としてveldraとrimuruを生成
・クラスで定義した変数は、インスタンスごとに用意されるので、インスタンス名.変数名 でアクセスできる
・クラスの中で定義したものをメンバといい、インスタンスの中の変数をインスタンス変数という
例: veldra.name = "ヴェルドラ"; veldra.hp = 100; rimuru.mp = 500;
※「.」はドット演算子ともいい、日本語の「の」に近い
・この仕組を用いるだけでも、変数のグループ化やデータの扱いの可視化が容易になる

p.156 simpleclass.cs

```
//p.156 simpleclass.cs
using System;
class myclass //クラスを定義(インスタンス変数を1個もつ部品的なクラス)
{
    public int x; //外部からアクセス可能なインスタンス変数
}
class simpleclass //実行用のMainメソッドを持つクラス
{
    public static void Main()
    {
        myclass mc = new myclass(); //部品的なクラスのインスタンスを生成
        mc.x = 10; //インスタンス名.メンバ名でインスタンス変数に数値を代入
        Console.WriteLine("mc.x = {0}", mc.x); //インスタンス名.メンバ名でインスタンス変数の値を表示
    }
}
```

アレンジ演習:p.156 simpleclass.cs

・上記の例から以下を試そう:
class Monster { public string name; public int hp; public int mp; } //名前とHP,MPを持つ怪物
Monster veldra = new Monster(); //怪物クラスから実体としてveldraを生成
veldra.name = "ヴェルドラ"; veldra.hp = 100;
・適当な表示処理を追加すること

作成例

```
//アレンジ演習:p.156 simpleclass.cs
using System;
class Monster //クラスを定義(インスタンス変数を3個もつ部品的なクラス)
{
    public string name; //外部からアクセス可能なインスタンス変数
    public int hp; //外部からアクセス可能なインスタンス変数
    public int mp; //外部からアクセス可能なインスタンス変数
}
class simpleclass //実行用のMainメソッドを持つクラス
{
    public static void Main()
    {
        Monster veldra = new Monster(); //部品的なクラスのインスタンスを生成
        veldra.name = "ヴェルドラ"; //インスタンス名.メンバ名でインスタンス変数に値(文字列)を代
入
        veldra.hp = 100; //インスタンス名.メンバ名でインスタンス変数に数値を代入
        //インスタンス名.メンバ名でインスタンス変数の値を表示(mpは初期値の0)
        Console.WriteLine("name = {0} hp = {1} mp = {2}", veldra.name,
veldra.hp, veldra.mp);
    }
}
```

p.157(複数のインスタンスを生成)

・メンバであるインスタンス変数は、インスタンスごとに存在する。よって、異なる変数として扱える
 例:
 Monster veldra = new Monster(), rimuru = new Monster(); //怪物クラスから実体として
veldraとrimuruを生成
 veldra.name = "ヴェルドラ"; veldra.hp = 100; veldra.mp = 500;
 rimuru.name = "魔王リムル"; rimuru.hp = 900; rimuru.mp = 800;

p.157 simpleclass02.cs

```
//p.157 simpleclass02.cs
using System;
class MyClass //クラスを定義(インスタンス変数を1個もつ部品的なクラス)
{
    public int x; //外部からアクセス可能なインスタンス変数
}
class simpleclass02
{
    public static void Main()
    {
        MyClass a, b; //部品的なクラスを型とするオブジェクト名を2つ宣言
        a = new MyClass(); //部品的なクラスのインスタンスを生成してオブジェクト名aで扱う
        b = new MyClass(); //部品的なクラスの別のインスタンスを生成してオブジェクト名bで扱う
        a.x =  10; //aに含まれるインスタンス変数xに代入
        b.x = 100; //bに含まれるインスタンス変数xに代入(↑とは別のもの)
        Console.WriteLine("a.x = {0}, b.x = {1}", a.x, b.x); //別のものなので10,100
となる
    }
}
```

アレンジ演習:p.157 simpleclass02.cs

・上記の例から以下を試そう:
```
  class Monster { public string name; public int hp; public int mp; } //名前とHP,MPを持つ怪物
 Monster veldra = new Monster(), rimuru = new Monster(); //怪物クラスから実体としてveldraとrimuruを生成
 veldra.name = "ヴェルドラ"; veldra.hp = 100; veldra.mp = 500;
 rimuru.name = "魔王リムル"; rimuru.hp = 900; rimuru.mp = 800;
```
・適当な表示処理を追加すること

作成例

```
//アレンジ演習:p.157 simpleclass02.cs
using System;
class Monster //クラスを定義(インスタンス変数を3個もつ部品的なクラス)
{
    public string name; //外部からアクセス可能なインスタンス変数
    public int hp; //外部からアクセス可能なインスタンス変数
    public int mp; //外部からアクセス可能なインスタンス変数
}
class simpleclass //実行用のMainメソッドを持つクラス
{
    public static void Main()
    {
        Monster veldra = new Monster(), rimuru = new Monster(); //部品的なクラスのインスタンスを2つ生成
        //インスタンス名.メンバ名でインスタンス変数に値(文字列、整数)を代入
        veldra.name = "ヴェルドラ"; veldra.hp = 100; veldra.mp = 500;
        rimuru.name = "魔王リムル"; rimuru.hp = 900; rimuru.mp = 800;
        //インスタンス名.メンバ名でインスタンス変数の値を表示
        Console.WriteLine("name = {0} hp = {1} mp = {2}", veldra.name, veldra.hp, veldra.mp);
        Console.WriteLine("name = {0} hp = {1} mp = {2}", rimuru.name, rimuru.hp, rimuru.mp);
    }
}
```

p.158(クラス型の変数の特性)

・クラスを型とする変数は参照型(p.40)となり、変数が持つのは参照(実体の位置情報)となる
・よって、初期化や生成により、変数にインスタンスの参照が与えられる
・そして、クラスを型とする変数どうしの代入を行うと、参照がコピーされる
・この結果、2つの変数が同じインスタンスを参照するようになる
(2つ目の変数が別名と同じ働きをするようになる)
例:
```
  Monster rimuru = new Monster(); //インスタンスの生成
  Monster maou = rimuru; //インスタンスの参照をコピー(maouがrimuruの別名になる)
  rimuru.hp = 10;
  Console.WriteLine(maou.hp); //10が表示される
```

p.158 simpleclass03.cs

```
//p.158 simpleclass03.cs
using System;
class MyClass //クラスを定義(インスタンス変数を1個もつ部品的なクラス)
{
    public int x; //外部からアクセス可能なインスタンス変数
}
class simpleclass03
{
    public static void Main()
    {
        MyClass a, b; //部品的なクラスを型とするオブジェクト名を2つ宣言
        a = new MyClass(); //部品的なクラスのインスタンスを生成してオブジェクト名aで扱う
        a.x = 10; //aに含まれるインスタンス変数xに代入
        b = a; //aが持つ参照をbにコピー(bはaと同じインスタンスを指すようになる)
        Console.WriteLine("b.x = {0}", b.x); //よってb経由でxの値が得られる
        b.x = 100; //b経由でxの値を変更でkる
        Console.WriteLine("a.x = {0}", a.x); //よってa経由でxの値を見ると変更されている
    }
}
```

提出:アレンジ演習:p.157 simpleclass02.cs

次回予告:p.159(Mainメソッドを含むクラス)