講義メモ
・p.159（Mainメソッドを含むクラス）から

p.159（Mainメソッドを含むクラス）

・simpleclass03などのように、インスタンス変数を持つクラスと、Mainメソッドを持つクラスは別にするのが基本
・そして、インスタンス変数を持つクラスはプログラム部品として設計・活用することが多い
・しかし、Mainメソッドを持つクラスにインスタンス変数を持つこともできる
・このようなクラスのインスタンスにはMainメソッドは含まれないので、インスタンス変数を持つクラスと、Mainメソッドを持つクラスを別にした場合と同様に動作する
・よって、例外的な記述法だが、次項以降で説明するメソッドのテスト処理の記述などに用いることもある

p.159 simpleclass04

```
//p.159 simpleclass04
using System;
class simpleclass04 {
    private int x; //インスタンス変数(このクラス内からのみ利用可能)
    public static void Main() {
        simpleclass04 s; //自分が属するクラスを型とする変数を定義できる
        s = new simpleclass04(); //自分が属するクラスのインスタンスを生成できる
        s.x = 10; //このインスタンスの中に、インスタンス変数があるので代入可能
        Console.WriteLine("s.x = " + s.x); //表示も可能
    }
}
```

p.160 メソッドを定義しよう

・C言語の構造体とは異なり、クラスには処理を行うメソッドを自由に定義できる
・なお、Mainメソッドは特別な意味をもつメソッドとして扱われる
・メソッドはC言語などの関数と同じ仕掛けで、0個以上の情報を受け取り、処理して、0または1個の情報を返すことができる
・返す値を戻り値（返却値）といい、メソッド定義の冒頭で、その型を示す。
・なお、返す値が0個の場合は、無を意味するvoidを指定する
・返す値が1個の場合は、メソッドの中に「return 式または値;」を記述し、どういうアルゴリズムになっていても、必ずreturnができないとコンパイルエラーになるので注意。
・なお、returnの後続部分は実行されないので、絶対に実行されない状態にしてしまうと、コンパイルエラーになる
・returnには式や他のメソッドの呼び出しが記述できる（メソッドの処理内容を全て記述できることもある）
・例：int add3(int a, int b, int c) { return a + b + c; }
・0個以上の情報を受け取る仕掛けが引数で、上のa,b,cのように、型と共に引数リスト(パラメータリスト)として、メソッド名直後のカッコ内に記述する
・0個の場合も、カッコは省略不可
※ メソッド側で定義している引数を仮引数ともいう（メソッドを呼び出す側で指定するのは実引数）
・メソッドのアクセス修飾子は、通常、private(利用自由)かpublic(外部からの利用不可)
・定義書式：アクセス修飾子 戻り値型 メソッド名(引数リスト) {…}
・例：

```
 public int add3(int a, int b, int c) { //利用自由でint型3引数をを受け取りint型の値を返すメソッド
   return a + b + c; //3引数の値の和を返す
 }
```

p.161（メソッドの呼び出し）

・インスタンス変数と同様に、インスタンスを生成すると呼び出し可能になるのが基本
・インスタンス変数と同様に「インスタンス名.メソッド名(引数リスト)」と呼び出すことが可能
・戻り値がvoidではない場合、戻り値を受け取って変数に格納したり、そのまま表示したりすることが可能
・例:

```
class Sub {
 public int add3(int a, int b, int c) { //利用自由でint型3引数ををを受け取りint型の値を返すメソッド
   return a + b + c; //3引数の値の和を返す
 }
}
class Center {
 :
 Sub s = new Sub();
 int sum = s.add3(1,2,3); //add3メソッドの戻り値を変数の初期化に用いることもOK
```

・通常のメソッドはインスタンス変数と同様に、インスタンス経由でアクセスされるので、インスタンスメソッドともいう
・メソッドに定義されている引数の数と型が呼び出しにおいてチェックされ、異なるとエラーになる
・なお、型が異なっても暗黙の変換が可能な場合はエラーにならない
・実引数と仮引数は名前は異なって良く、メソッドの汎用性を考えると異なる名前の方が良い
・実引数と仮引数は定義順に割り当てられる
・実引数の代わりに、値を直接記述しても良い

p.161 method01.cs

```
//p.161 method01.cs
using System;
class MyClass { //メソッドを持つクラス
    public int Add(int x, int y) { //外部利用可能で戻り値型がintで引数がint型2値のAddメソッドの定義
        int z; //メソッド内部で定義されているのでローカル変数
        z = x + y;
        return z; //整数の和が格納された変数を指定することで、その値が返される
    }
}
class method01 {
    public static void Main() {
        MyClass a = new MyClass(); //メソッドを持つクラスのインスタンスを生成
        int sum; //合計用
        sum = a.Add(100, 200); //インスタンス経由でメソッドを呼び、整数2値を渡し、戻り値を変数に代入
        Console.WriteLine("sum = {0}", sum);
    }
}
```

アレンジ演習:p.161 method01.cs

・メソッドに渡す整数2値を(100, 200固定ではなく)コンソールから入力できるようにしよう

作成例

```
//アレンジ演習:p.161 method01.cs
using System;
class MyClass { //メソッドを持つクラス
    public int Add(int x, int y) { //外部利用可能で戻り値型がintで引数がint型2値のAddメソッドの定義
```

```
        int z; //メソッド内部で定義されているのでローカル変数
        z = x + y;
        return z; //整数の和が格納された変数を指定することで、その値が返される
    }
}
class method01 {
    public static void Main() {
        MyClass a = new MyClass(); //メソッドを持つクラスのインスタンスを生成
        int sum; //合計用
        Console.Write("x:"); int x = int.Parse(Console.ReadLine()); //【追加】コンソールから入力
        Console.Write("y:"); int y = int.Parse(Console.ReadLine()); //【追加】コンソールから入力
        sum = a.Add(x, y); //【変更】インスタンス経由でメソッドを呼び、整数2値を渡し、戻り値を変数に代入
        Console.WriteLine("sum = {0}", sum);
    }
}
```

p.163 bmiclass.cs

```
//p.163 bmiclass.cs
using System;
class BMI { //BMI計算用の部品クラス
    private double blm; //身長(m単位):外部から見えないインスタンス変数
    public double Calc(double bl, double bw) { //利用自由なインスタンスメソッド(戻り値有、引数2値)
        blm = bl / 100.0; //引数blで受け取った身長(cm単位)をメートル単位に換算
        return bw / Math.Pow(blm, 2.0); //BMIを計算して(体重÷身長の2乗)返す
    }
}
class bmiclass {
    public static void Main() {
        string strBl, strBw; //入力用
        double blcm, bwkg; //変換結果
        Console.Write("身長(cm)---");
        strBl = Console.ReadLine();
        blcm = Double.Parse(strBl);
        Console.Write("体重(kg)---");
        strBw = Console.ReadLine();
        bwkg = Double.Parse(strBw);
        BMI bmi = new BMI(); //Calcメソッドを持つBMIクラスのインスタンスを生成
        Console.WriteLine("BMIは{0:#.##}です", bmi.Calc(blcm, bwkg)); //メソッドを呼んで戻り値を表示
    }
}
```

p.164(値を返さないメソッド)

・値を返さないメソッドでは、戻り値型をvoidとする(省略不可)
・値を返さないメソッドには、returnは記述不要(記述しても良い)
・なお、メソッドを途中にreturnを記述して、途中で打ち切ることも可能だが、実行されることがない文ができてしまうとエラーになる
・OKな例:

```
 void foo(int a) {
  if (a == 0) { return; } //0の時はここで打ち切ってOK
  : //これ以降は0以外の時に実行されるのでOK
 }
```
・NGな例：
```
 void foo(int a) {
  :
  return; //どんな場合もここで打ち切ってOK
  : //これ以降は実行されないのでNG
 }
```
・なお、p.165 noreturn.csのすべてのreturnは省略可能だが、記述しても良い
・値を返さないメソッドのメソッド末尾のreturnは省略することが多い
・メソッド途中のreturnは省略可能な場合でも省略しないことがある（保守性が上がる）

p.165 noreturnvalue.cs

```
//p.165 noreturnvalue.cs
using System;
class Kakeibo {
    private int total = 0; //残高：外部から見えないインスタンス変数
    public void nyukin(int en) { //入金：戻り値のないインスタンスメソッド
        total += en;
        Console.WriteLine("{0}円を入金しました", en);
        // return; //この後はないので省略可能
    }
    public void shishutsu(int en) { //支出：戻り値のないインスタンスメソッド
        if (total < en) { //残高不足？
            Console.WriteLine("{0}円も支出できません", en);
            // return; //この後にelseしかないので省略可能
        } else {
            total -= en;
            Console.WriteLine("{0}円を支出しました", en);
            // return; //この後はelseの終わりしかないので省略可能
        }
    }
    public void gettotal() { //残高照会：戻り値のないインスタンスメソッド
        if (total == 0) {
            Console.WriteLine("残高はありません");
            // return; //この後にelseしかないので省略可能
        } else {
            Console.WriteLine("残高は{0}円です", total);
            // return; //この後はelseの終わりしかないので省略可能
        }
    }
}
class noreturnvalue {
    public static void Main() {
        Kakeibo k = new Kakeibo();
        k.gettotal(); //残高照会⇒残高はありません
        k.nyukin(1000); //入金1000円
        k.gettotal(); //残高照会⇒1000円です
        k.nyukin(2000); //入金2000円
        k.gettotal(); //残高照会⇒3000円です
        k.shishutsu(500); //支出500円
```

```
        k.gettotal(); //残高照会⇒2500円です
        k.shishutsu(10000); //支出10000円⇒支出できません
        k.gettotal(); //残高照会⇒2500円です
    }
}
```

p.166（カプセル化）

・インスタンス変数のうち、外部から直接アクセスする必要がないものや、させない方が良いものはprivate指定すると良い
・外部から直接アクセスさせないようにprivate指定して変数値が適切な状態に保つことをカプセル化という
・例えば、身長や体重を正の数に制限したり、点数を0から100までに制限する場合などに用いる
・この場合、必要に応じてprivate指定のインスタンス変数にアクセスするメソッドを記述し、この中で、アクセス内容を制限する
例：

```
 private double bw; //体重：外部から見えないインスタンス変数
 public void setBw(double w) { if (w > 0) {bw = w;} } //引数で体重を受け取って正の数なら代入
 public double getBw() { return bw;} //体重を返す
```

※ このようなメソッドをアクセッサといい、代入用をセッター、参照用をゲッターともいう
※ なお、C#には、アクセッサの実装に便利なプロパティ(p.207)という仕掛けがあり、プロパティを用いるのが基本

p.167 コンストラクタ

・new等でインスタンスの生成を行う時に、同時に実行したい処理を特殊な書式で記述することができる
・これがコンストラクタで、省略すると、自動的に中身がないコンストラクタが用意されて用いられる
・コンストラクタ＝構築なので、生成時に呼ばれる特殊なメソッドを指す
・コンストラクタを自前で記述することで、インスタンスの生成を行う時に実行したいことを書ける
・主に、データメンバの初期化や、開始処理などの記述に用いて「必ず実行させる」ことが可能
・なお、コンストラクタには名前はなく、クラス名で記述し、public指定で、戻り値はない
・主な書式： public クラス名() {…}
・「new クラス名()」で呼び出して欲しい場合はカッコ内は空にする
 ※ 引数を指定することも可能で、引数のない定義と両立できる（後述：オーバーロード）

p.167 construct01.cs

```
//p.167 construct01.cs
using System;
class MyClass {
    int x; //外部から見えないインスタンス変数
    public void showx() { //表示：インスタンスメソッド
        Console.WriteLine("x = " + x);
    }
    public MyClass() { //コンストラクタ
        x = 10; //インスタンス変数の初期化
        Console.WriteLine("xに10を代入しました");
    }
}
class construct01 {
    public static void Main() {
        MyClass mc = new MyClass(); //インスタンスの生成と共にコンストラクタを実行
        mc.showx(); //インスタンスメソッドを実行
    }
```

```
}
```

アレンジ演習：p.165 noreturnvalue.cs

・残高を0で初期化する処理をコンストラクタで行うようにしよう
・実行結果は変わらないので、テスト用にコンストラクタに「初期化しました」と表示する処理を加えよう

作成例

```
//アレンジ演習：p.165 noreturnvalue.cs
using System;
class Kakeibo {
    private int total; //【変更】残高：外部から見えないインスタンス変数
    public Kakeibo() { //【以下追加】コンストラクタ
        total = 0; //インスタンス変数を初期化
        Console.WriteLine("初期化しました"); //テスト用
    }
    public void nyukin(int en) { //入金：戻り値のないインスタンスメソッド
        total += en;
        Console.WriteLine("{0}円を入金しました", en);
        // return; //この後はないので省略可能
    }
    public void shishutsu(int en) { //支出：戻り値のないインスタンスメソッド
        if (total < en) { //残高不足？
            Console.WriteLine("{0}円も支出できません", en);
            // return; //この後にelseしかないので省略可能
        } else {
            total -= en;
            Console.WriteLine("{0}円を支出しました", en);
            // return; //この後はelseの終わりしかないので省略可能
        }
    }
    public void gettotal() { //残高照会：戻り値のないインスタンスメソッド
        if (total == 0) {
            Console.WriteLine("残高はありません");
            // return; //この後にelseしかないので省略可能
        } else {
            Console.WriteLine("残高は{0}円です", total);
            // return; //この後はelseの終わりしかないので省略可能
        }
    }
}
class noreturnvalue {
    public static void Main() {
        Kakeibo k = new Kakeibo();
        k.gettotal(); //残高照会⇒残高はありません
        k.nyukin(1000); //入金1000円
        k.gettotal(); //残高照会⇒1000円です
        k.nyukin(2000); //入金2000円
        k.gettotal(); //残高照会⇒3000円です
        k.shishutsu(500); //支出500円
        k.gettotal(); //残高照会⇒2500円です
        k.shishutsu(10000); //支出10000円⇒支出できません
        k.gettotal(); //残高照会⇒2500円です
```

```
    }
}
```

p.168（引数のあるコンストラクタ）

・コンストラクタに引数リストを記述することができる
・これにより、インスタンス変数の初期値を引数で渡すことが多い
・例：

```
 public Kakeibo(int t) { //引数のあるコンストラクタ
  total = t; //インスタンス変数を引数の値で初期化
 }
```

・引数のあるコンストラクタを呼び出すには、newのカッコ内に引数の値や式を記述する
・例：

```
 Kakeibo k = new Kakeibo(10000); //インスタンスを生成し、残高を10000円とする
```

アレンジ演習：p.165 noreturnvalue.cs

・残高を引数tで初期化する処理をコンストラクタで行うようにしよう
・上記の例の通り、インスタンスを生成し、残高を10000円としよう

作成例

```
//アレンジ演習：p.165 noreturnvalue.cs
using System;
class Kakeibo {
    private int total; //残高：外部から見えないインスタンス変数
    public Kakeibo(int t) { //【変更】コンストラクタ
        total = t; //【変更】インスタンス変数を引数で初期化
        Console.WriteLine("初期化しました"); //テスト用
    }
    public void nyukin(int en) { //入金：戻り値のないインスタンスメソッド
        total += en;
        Console.WriteLine("{0}円を入金しました", en);
        // return; //この後はないので省略可能
    }
    public void shishutsu(int en) { //支出：戻り値のないインスタンスメソッド
        if (total < en) { //残高不足？
            Console.WriteLine("{0}円も支出できません", en);
            // return; //この後にelseしかないので省略可能
        } else {
            total -= en;
            Console.WriteLine("{0}円を支出しました", en);
            // return; //この後はelseの終わりしかないので省略可能
        }
    }
    public void gettotal() { //残高照会：戻り値のないインスタンスメソッド
        if (total == 0) {
            Console.WriteLine("残高はありません");
            // return; //この後にelseしかないので省略可能
        } else {
            Console.WriteLine("残高は{0}円です", total);
            // return; //この後はelseの終わりしかないので省略可能
        }
    }
```

```
}
class noreturnvalue {
    public static void Main() {
        Kakeibo k = new Kakeibo(10000); //【変更】引数で残高の初期値を与える
        k.gettotal(); //残高照会
        k.nyukin(1000); //入金
        k.gettotal(); //残高照会
        k.nyukin(2000); //入金
        k.gettotal(); //残高照会
        k.shishutsu(500); //支出
        k.gettotal(); //残高照会
        k.shishutsu(10000); //支出
        k.gettotal(); //残高照会
    }
}
```

p.168（コンストラクタのオーバーロード）

・引数のないコンストラクタと、引数のあるコンストラクタを併記できる
・newにおいて引数が指定されたかどうかによって、自動的に使い分けが行われる
・また、この時に、引数とコンストラクタの引数リストのチェックが行われ、引数の個数と型が一致しないとエラーになる
・このチェックがあるので、引数のあるコンストラクタを複数併記でき、呼び出しを一致するものが利用される
・この仕掛けをオーバーロード（多重定義）という
・また、チェックに用いる「引数の個数と型の並び」をシグニチャという
・定義例：

```
 public Kakeibo() { //引数の無いコンストラクタ
  total = 0; //インスタンス変数を0で初期化
 }
 public Kakeibo(int t) { //int型引数1個のあるコンストラクタ①
  total = t; //インスタンス変数を引数の値で初期化
 }
 public Kakeibo(int t, int u) { //int型引数2個のあるコンストラクタ②
  total = t; //インスタンス変数を引数の値で初期化
  count = u; //同上
 }
```

・呼び出し例

```
 Kakeibo n = new Kakeibo(); //インスタンスを生成し、引数の無いコンストラクタを呼ぶ
 Kakeibo k = new Kakeibo(10000); //インスタンスを生成し、コンストラクタ①を呼ぶ
 Kakeibo m = new Kakeibo(20000, 3); //インスタンスを生成し、コンストラクタ②を呼ぶ
```

・型の並びでも区別されるので、例えば、public Kakeibo(int t, double d)と、public Kakeibo(double d, int t)は併記可能

p.169 補足：construct02.csにおけるToString()について

・ToString()は一種のメソッドで、int.Parse()と同様に、intに対して指定でき、整数を文字列にした結果を返す
※ どちらも内部的には.NET型のSystem.Int32構造体(p.275)が持つオーバライドメソッド(p.230)

p.169 construct02.cs

```
//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;
    }
}
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 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

提出:アレンジ演習:p.169 construct02.cs

次回予告:p.171「デストラクタ」から再開します