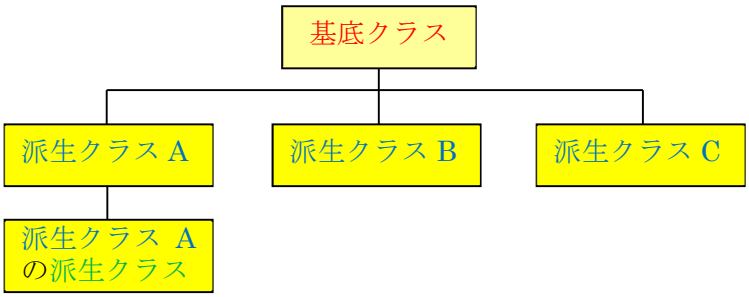
2023 年 6 月 29 日（木）実施

クラスの継承

オブジェクト指向プログラミングの基本的な属性として，親クラスのメンバを再利用，拡張，または変更する子クラスを定義することが出来る。 メンバの再利用を**継承**と呼び，継承元となるクラスを**基底クラス**と呼ぶ。また，基底クラスのメンバを継承するクラスを，**派生クラス**と呼ぶ。なお，メンバの中でコンストラクタは継承されない。

C#言語では，Java 言語と同様に**単一継承**のみがサポートされている。単一継承では，ある基底クラスから複数の派生クラスを作ることは可能であるが，ある派生クラスを複数の基底クラスから作ることは出来ない。（下図）



派生クラスの構成は一般に次の様になる。

```
namespace プロジェクト名
{
    修飾子 class 派生クラス名 : 基底クラス名
    {
        フィールド
        修飾子 コンストラクタ名(=派生クラス名)( 引数リスト )
        {
            初期化処理
        }
        修飾子 データ型 メソッド名 ( 引数リスト )
        {
            処理
        }
    }
}
```

オーバーライド

派生クラスで基底クラスから継承したメソッドを再定義することを**オーバーライド**（override）

と呼ぶ。C#言語では，メソッドをオーバーライドするためには，基底クラスのメソッドに virtual のキーワードを付けておいて，派生クラスでオーバーライドするメソッドには override キーワードを付ける必要がある。

カプセル化

C#言語では，アクセス修飾子によって，全てのクラスおよびクラスメンバに対して，他のクラスに提供するアクセスレベルを指定出来る。主なアクセス修飾子は次の様な働きをする。

・public　制限は無く，どこからでもアクセス出来る。
・private　同じクラス内でのみアクセス出来る。
・protected　同じクラス内または派生クラス内でのみアクセス出来る。
・internal　同じアセンブリ（アプリケーションのビルドの基本単位）内でのみアクセス出来る。

これらのアクセス修飾子の使い分けによって，外部からの影響を防ぎたいメンバを隠し，外部からの操作を許すメンバを公開してアクセス可能にすることを**カプセル化**と呼ぶ。

本日の課題

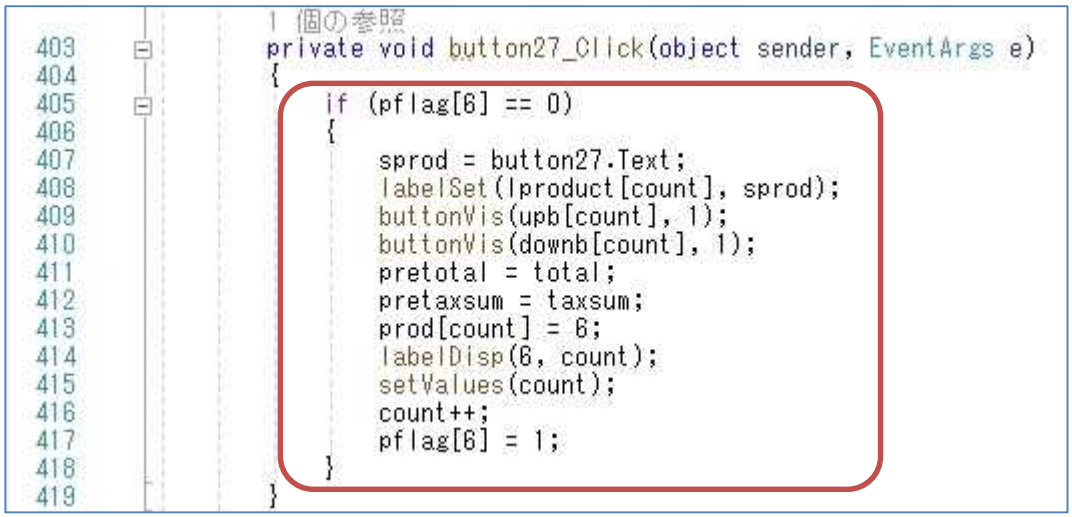
今回の授業では，レジスタアプリの作成を通じて，クラスの継承及びメンバのカプセル化について学ぶ。

手順

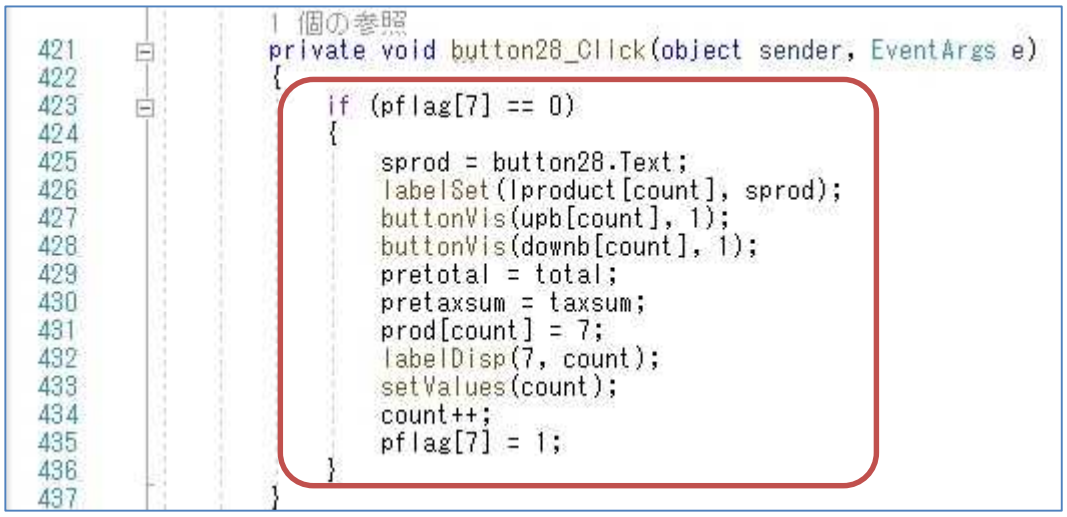
1）コーディング（前回の続き）

Form1 のフォームデザイナー上で button26～button28 をダブルクリックして，イベントハンドラ button26_Click～button28_Click の処理内容（赤枠の部分）を記述する。

```
1 個の参照
385     private void button26_Click(object sender, EventArgs e)
386     {
387         if (pflag[5] == 0)
388         {
389             sprod = button26.Text;
390             labelSet(lproduct[count], sprod);
391             buttonVis(upb[count], 1);
392             buttonVis(downb[count], 1);
393             pretotal = total;
394             pretaxsum = taxsum;
395             prod[count] = 5;
396             labelDisp(5, count);
397             setValues(count);
398             count++;
399             pflag[5] = 1;
400         }
401     }
```



```
1 個の参照
private void button27_Click(object sender, EventArgs e)
{
    if (pflag[6] == 0)
    {
        sprod = button27.Text;
        labelSet(lproduct[count], sprod);
        buttonVis(upb[count], 1);
        buttonVis(downb[count], 1);
        pretotal = total;
        pretaxsum = taxsum;
        prod[count] = 6;
        labelDisp(6, count);
        setValues(count);
        count++;
        pflag[6] = 1;
    }
}
```

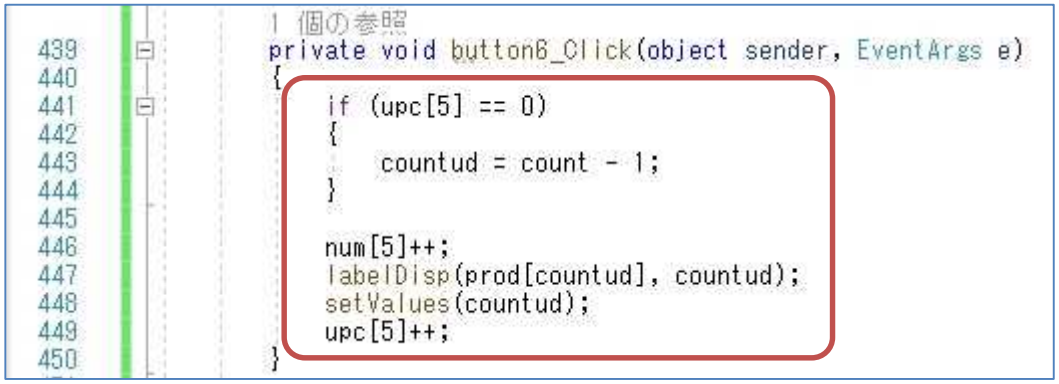


```
1 個の参照
private void button28_Click(object sender, EventArgs e)
{
    if (pflag[7] == 0)
    {
        sprod = button28.Text;
        labelSet(lproduct[count], sprod);
        buttonVis(upb[count], 1);
        buttonVis(downb[count], 1);
        pretotal = total;
        pretaxsum = taxsum;
        prod[count] = 7;
        labelDisp(7, count);
        setValues(count);
        count++;
        pflag[7] = 1;
    }
}
```

更に，アップボタン，ダウンボタンのイベントハンドラを作成する。

先ず，Form1 のフォームデザイナー上で button6～button8 をダブルクリックして，イベントハンドラ button6_Click～button8_Click の処理内容を次の様に記述する。

続いて，フォームデザイナー上で button16～button18 をダブルクリックして，イベントハンドラ button16_Click～button18_Click の処理内容を次ページの様に記述する。



```
1 個の参照
private void button6_Click(object sender, EventArgs e)
{
    if (upc[5] == 0)
    {
        countud = count - 1;
    }

    num[5]++;
    labelDisp(prod[countud], countud);
    setValues(countud);
    upc[5]++;
}
```

```
452         1 個の参照
            private void button7_Click(object sender, EventArgs e)
453         {
454             if (upc[6] == 0)
455             {
456                 countud = count - 1;
457             }
458
459             num[6]++;
460             labelDisp(prod[countud], countud);
461             setValues(countud);
462             upc[6]++;
463         }
464
            1 個の参照
465         private void button8_Click(object sender, EventArgs e)
466         {
467             if (upc[7] == 0)
468             {
469                 countud = count - 1;
470             }
471
472             num[7]++;
473             labelDisp(prod[countud], countud);
474             setValues(countud);
475             upc[7]++;
476         }
```

```
            1 個の参照
478         private void button16_Click(object sender, EventArgs e)
479         {
480             if (upc[5] > downc[5])
481             {
482                 num[5]--;
483                 labelDisp(prod[countud], countud);
484                 setValues(countud);
485                 downc[5]++;
486             }
487         }
488
            1 個の参照
489         private void button17_Click(object sender, EventArgs e)
490         {
491             if (upc[6] > downc[6])
492             {
493                 num[6]--;
494                 labelDisp(prod[countud], countud);
495                 setValues(countud);
496                 downc[6]++;
497             }
498         }
499
            1 個の参照
500         private void button18_Click(object sender, EventArgs e)
501         {
502             if (upc[7] > downc[7])
503             {
504                 num[7]--;
505                 labelDisp(prod[countud], countud);
506                 setValues(countud);
507                 downc[7]++;
508             }
509         }
```

2) クラスの作成及びコーディング 1

先ず,『ソリューションエクスプローラー』で『JimboRegister』を右クリックして，表示されるメニューで『追加』を選択し，更に表示されるメニューで『クラス』を選択する。そこで表示されるダイアログで,『C# 個の項目』及び『クラス』を選択し，「DispReceipt.cs」と入力して,『追加』ボタンをクリックする。

次に, 作成された DispReceipt クラスにフィールド, コンストラクタ及びメソッドを記述する。ここで，SetString メソッドはメッセージボックスに表示する文字列を生成するためのもので, receipt, total, taxsum を引数として受け取り, 結合された文字列を result に代入する。protected 修飾子があるので, Form2 クラスからは SetString メソッドに直接アクセスすることは出来ない。DispMB メソッドはメッセージボックスに SetString メソッドで得られた文字列を表示する。

DispReceipt.cs | ManageForm.cs | Form2.cs | Form2.cs [デザイン] | Form1.cs

JimboRegister　　JimboRegister.DispReceipt

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6   using System.Windows.Forms;
7
8   namespace JimboRegister
9   {
        4 個の参照
10      class DispReceipt
11      {
12          int total;
13          int taxsum;
14          string receipt;
15          public string result;
16
            2 個の参照
17          public DispReceipt(string s1, string s2, string s3)
18          {
19              receipt = s1;
20              total = Int32.Parse(s2);
21              taxsum = Int32.Parse(s3);
22
23              SetString(receipt, total, taxsum);
24          }
25
            1 個の参照
26          protected void SetString(string s, int t1, int t2)
27          {
28              result = s + "合計 " + String.Format("{0:#,0} 円¥n", t1)
29                  + " (税額 " + String.Format("{0:#,0} 円", t2) + ") ";
30          }
31
            2 個の参照
32          public virtual void DispMB()
33          {
34              MessageBox.Show(result);
35          }
36      }
37  }
```

手で入力する

続いて，Form2.cs のイベントハンドラ button1_Click の内容を変更して，DispReceipt クラス

のインスタンスを生成する。

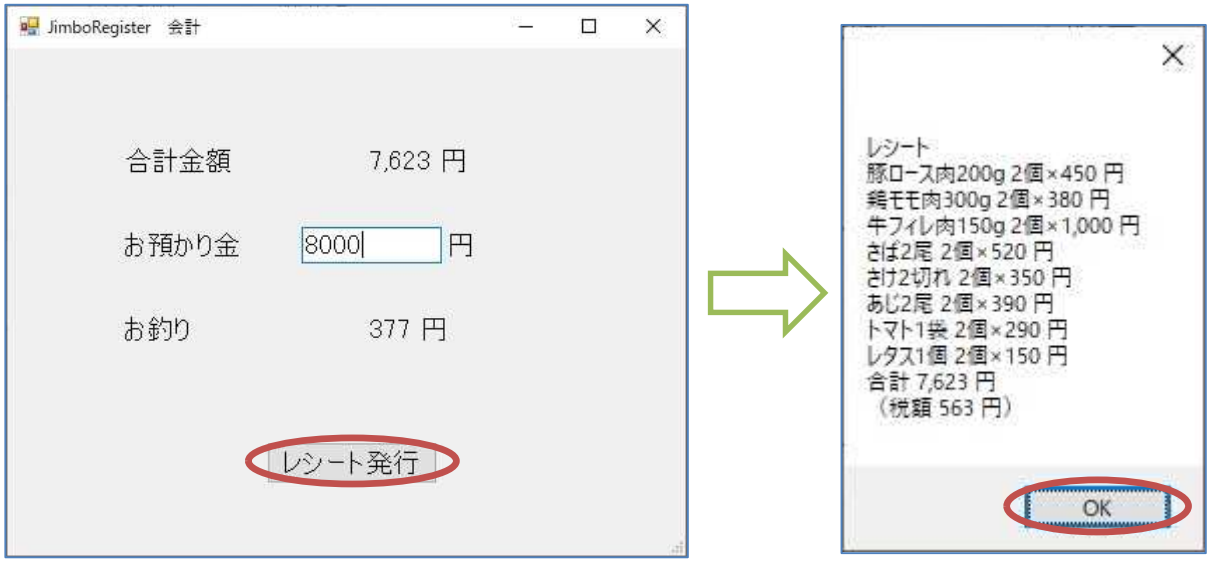
```
52      1 個の参照
        private void button1_Click(object sender, EventArgs e)
53      {
54          DispReceipt ds = new DispReceipt(receipt, total.ToString(), taxsum.ToString());
55          ds.DispMB();
56
57          //  MessageBox.Show(receipt + "合計 " + String.Format("{0:#,0} 円¥n", total)
58          //      + " (税額 " + String.Format("{0:#,0} 円", taxsum) + ") ");
59      }
```

3) 実行（途中経過）

ここで，『保存』ボタンを押してから，『開始』ボタンを押して，プログラムを実行する。

先ず，それぞれの商品ボタンを押してからアップボタン及びダウンボタンの動作を確かめるという操作を 8 件分行い，『会計』ボタンをクリックする。

次に，『お預かり金』の金額をテキストボックスに入力する。



4）クラスの作成及びコーディング 2

先ず、DispReceipt クラスを基底クラスとする派生クラス DispReceipt2 を作成する（ファイル名は DispReceipt2.cs）。作成された DispReceipt2 クラスにフィールド，コンストラクタ及びメソッドを記述する。ここで，cash は受け取った現金の金額を格納する変数である。また，DispMB メソッドは基底クラスでの定義をオーバーライドにより再定義している。

```
DispReceipt2.cs | DispReceipt.cs | Form2.cs | Form2.cs [デザイン] | Form1.cs | Form1.cs [デザイン]
JimboRegister | JimboRegister.DispReceipt2 | result2
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6  using System.Windows.Forms;
 7
 8  namespace JimboRegister
 9  {
        3 個の参照
10      class DispReceipt2 : DispReceipt
11      {
12          int cash;
13          int change;
14          string result2;
15
            1 個の参照
16          public DispReceipt2(string s1, string s2, string s3, string s4, string s5)
17              : base(s1, s2, s3)
18          {
19              cash = Int32.Parse(s4);
20              change = Int32.Parse(s5);
21              result2 = result;
22          }
23
            2 個の参照
24          public override void DispMB()
25          {
26              result2 += "¥n現金 " + String.Format("{0:#,0} 円¥n", cash)
27                  + "（お釣り " + String.Format("{0:#,0} 円", change) + "）";
28              MessageBox.Show(result2);
29          }
30      }
31  }
```

コンストラクタ DispReceipt2 は 5 個の引数を有するが，：base(s1, s2, s3) により，引数 s1, s2，s3 を引き渡して，基底クラスのコンストラクタを呼び出している。また，result は基底クラスのフィールド変数 result を参照している。これは，基底クラスで result の宣言の際に public 修飾子を付けていることで実現される。クラスメンバの修飾子を省略した場合には，private のアクセスレベルとなることに注意が必要である。

次に，DispReceipt2 クラスを利用するために，以下の様に Form2.cs を書き換える。

① フィールド変数として，cash 及び change を宣言する。

```
13  public partial class Form2 : Form
14  {
15      string[] arg;
16      int total;
17      int taxsum;
18      int cash;
19      int change;
20      string receipt;
21
```

② イベントハンドラ textBox1_TextChanged の処理内容を次の様に変更する。

```
        1 個の参照
42      private void textBox1_TextChanged(object sender, EventArgs e)
43      {
44          cash = Int32.Parse(textBox1.Text);
45          change = cash - total;
46
47          if (change >= 0)
48          {
49              label6.Text = String.Format("{0:#,0} 円", change);
50          }
51      }
```

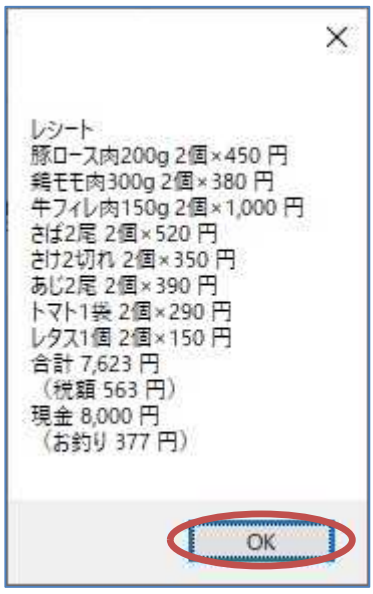
③ イベントハンドラ button1_click の処理内容を次の様に変更する（教材の行幅の都合上，5 個の引数を並べる際に折り返している。）。

```
        1 個の参照
53      private void button1_Click(object sender, EventArgs e)
54      {
55          //DispReceipt ds = new DispReceipt(receipt, total.ToString(), taxsum.ToString());
56          //ds.DispMB();
57
58          DispReceipt2 ds2 = new DispReceipt2(receipt, total.ToString(), taxsum.ToString(),
59              cash.ToString(), change.ToString());
60          ds2.DispMB();
61
62          //    MessageBox.Show(receipt + "合計 " + String.Format("{0:#,0} 円¥n", total)
63          //        + " (税額 " + String.Format("{0:#,0} 円", taxsum) + ") ");
64      }
```

5）実行（最終結果）

ここで，『保存』ボタンを押してから，『開始』ボタンを押して，プログラムを実行する。

最初のフォームで会計ボタンを押し，2 番目のフォームでお預かり金額を入力するところまではこれまで通りの動作であるが，レシート発行ボタンを押した後のレシートの内容が変化することを確認する。



提出物：

1）コードエディタで編集したソースファイル Form1.cs をメールに添付して提出する。

2）コードエディタで編集したソースファイル Form2.cs をメールに添付して提出する。

3）DispReceipt クラスのソースファイル DispReceipt.cs をメールに添付して提出する。

4）DispReceipt2 クラスのソースファイル DispReceipt2.cs をメールに添付して提出する。

5）完成後に実行して，アプリが起動後，8 件分の商品登録及びアップボタン及びダウンボタンの動作を確かめた状態のスクリーンショット第 10 回実行結果.jpg（.png も可）をメールに添付して提出する。

6）上の状態の後，『会計』ボタンをクリックして，表示されたフォームのテキストボックスに『お預かり金』の金額を入力した状態のスクリーンショット第 10 回会計.jpg（.png も可）をメールに添付して提出する。

7）上の状態の後，『レシート発行』ボタンをクリックして表示されたメッセージボックスのスクリーンショット第 10 回レシート.jpg（.png も可）をメールに添付して提出する。

8）質問を記述したファイル Prog2_Questions_10th.txt に解答を書き込んで保存し，メールに添付して提出する。