2023 年 6 月 22 日（木）実施

クラスの基礎

第 1 回の教材では，Form1.cs を例にとってクラスの構成要素について解説した。Visual Studio で C#言語によるフォームアプリケーションを作成する際，プロジェクトを新規作成すると自動的に生成される Program.cs は次の様になっている。

【Program.cs】

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace JimboRegister
{
    static class Program
    {
        /// <summary>
        /// アプリケーションのメイン エントリ ポイントです。
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

プログラムが実行される際には，Program クラスの Main メソッドが最初に実行される。説明は省くが，Main メソッドの最終行の機能は，現在のスレッドで標準のアプリケーションメッセージループの実行を開始し，指定したフォームを表示するものである。ここで，new 演算子によって，Form1 クラスの**インスタンス**（実体）が生成されるが，new 演算子は新しいインスタンスをメモリ上に割り当て，**コンストラクタ**を呼び出してインスタンスを初期化し，インスタンスへの参照を返す。

Form1.cs には，フォームを定義する Form1 クラスが必須であるが，それ以外のクラスを作成して含めることが出来る。また，クラスは別のファイルに作成することも出来る。クラスの構成は一般に次の様になる。

（図は次のページ）

```
namespace プロジェクト名
{
    修飾子 class クラス名
    {
        フィールド

        修飾子 コンストラクタ名( 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
        }
    }
}
```

コンストラクタ名はクラス名と同一

本日の課題

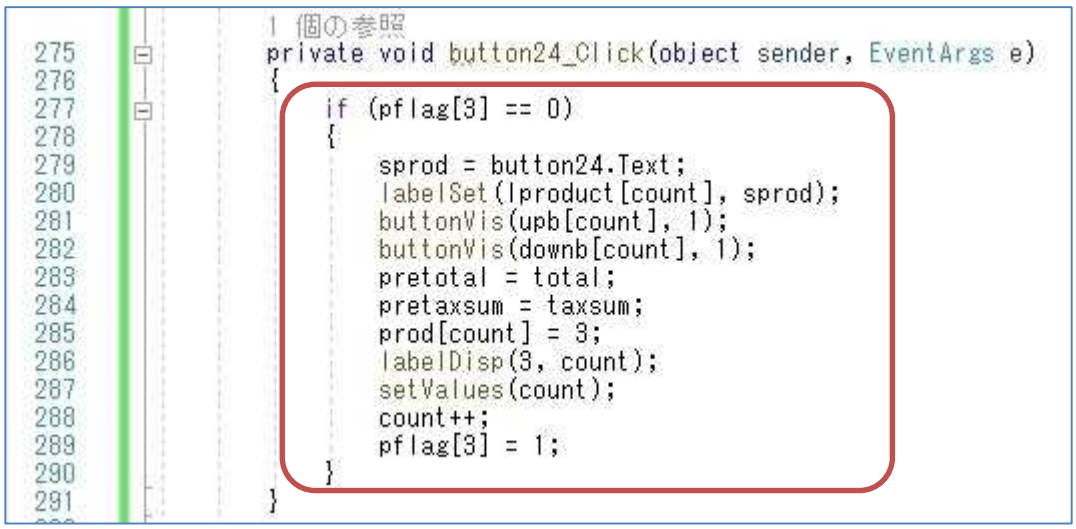
今回の授業では，レジスタアプリの作成を通じて，クラスの作成及びインスタンスの生成について学ぶ。

手順

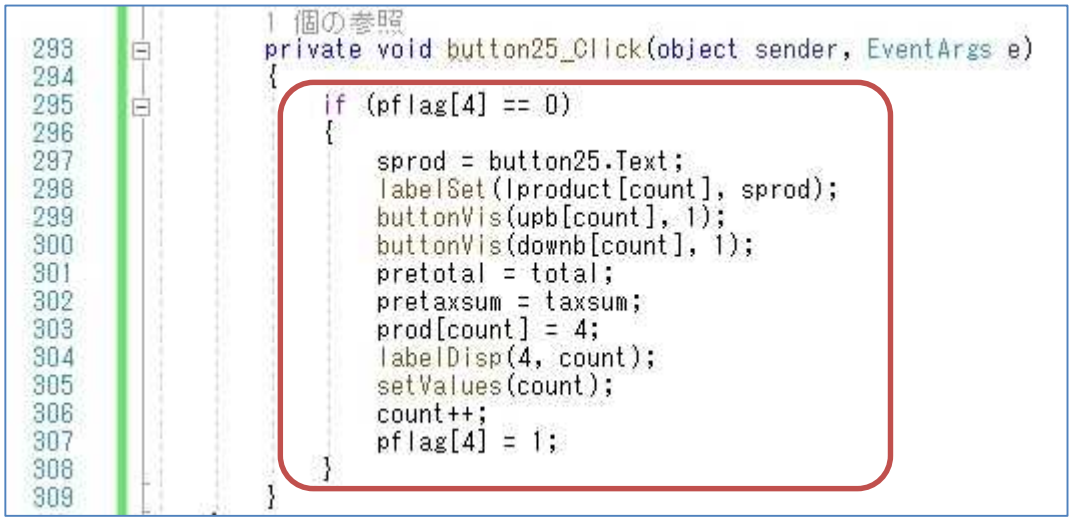
1）コーディング（前回の続き）

フォームデザイナー上で button23～button25 をダブルクリックして，イベントハンドラ button23_Click～button25_Click の処理内容（赤枠の部分）を記述する。

```
1 個の参照
257     private void button23_Click(object sender, EventArgs e)
258     {
259         if (pflag[2] == 0)
260         {
261             sprod = button23.Text;
262             labelSet(lproduct[count], sprod);
263             buttonVis(upb[count], 1);
264             buttonVis(downb[count], 1);
265             pretotal = total;
266             pretaxsum = taxsum;
267             prod[count] = 2;
268             labelDisp(2, count);
269             setValues(count);
270             count++;
271             pflag[2] = 1;
272         }
273     }
```



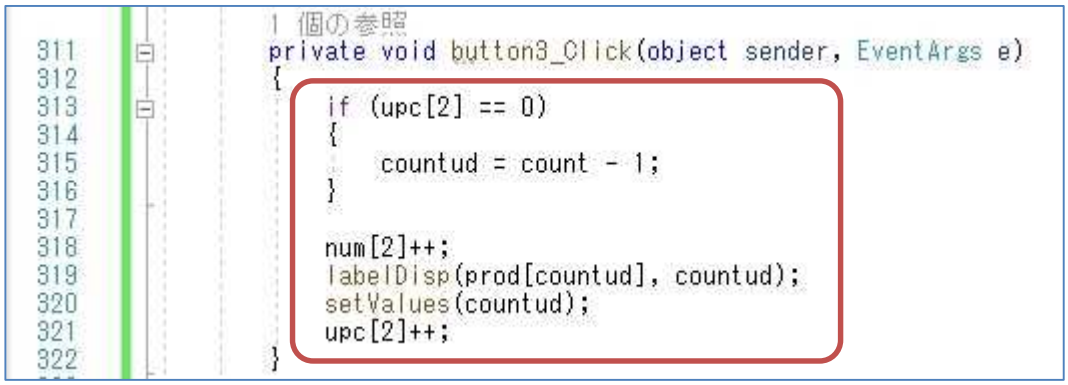
```
1 個の参照
private void button24_Click(object sender, EventArgs e)
{
    if (pflag[3] == 0)
    {
        sprod = button24.Text;
        labelSet(lproduct[count], sprod);
        buttonVis(upb[count], 1);
        buttonVis(downb[count], 1);
        pretotal = total;
        pretaxsum = taxsum;
        prod[count] = 3;
        labelDisp(3, count);
        setValues(count);
        count++;
        pflag[3] = 1;
    }
}
```



```
1 個の参照
private void button25_Click(object sender, EventArgs e)
{
    if (pflag[4] == 0)
    {
        sprod = button25.Text;
        labelSet(lproduct[count], sprod);
        buttonVis(upb[count], 1);
        buttonVis(downb[count], 1);
        pretotal = total;
        pretaxsum = taxsum;
        prod[count] = 4;
        labelDisp(4, count);
        setValues(count);
        count++;
        pflag[4] = 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，フォームデザイナー上で button3～button5 をダブルクリックして，イベントハンドラ button3_Click～button5_Click の処理内容を次の様に記述する。



```
1 個の参照
private void button3_Click(object sender, EventArgs e)
{
    if (upc[2] == 0)
    {
        countud = count - 1;
    }

    num[2]++;
    labelDisp(prod[countud], countud);
    setValues(countud);
    upc[2]++;
}
```

```
        1 個の参照
        private void button4_Click(object sender, EventArgs e)
        {
            if (upc[3] == 0)
            {
                countud = count - 1;
            }

            num[3]++;
            labelDisp(prod[countud], countud);
            setValues(countud);
            upc[3]++;
        }

        1 個の参照
        private void button5_Click(object sender, EventArgs e)
        {
            if (upc[4] == 0)
            {
                countud = count - 1;
            }

            num[4]++;
            labelDisp(prod[countud], countud);
            setValues(countud);
            upc[4]++;
        }
```

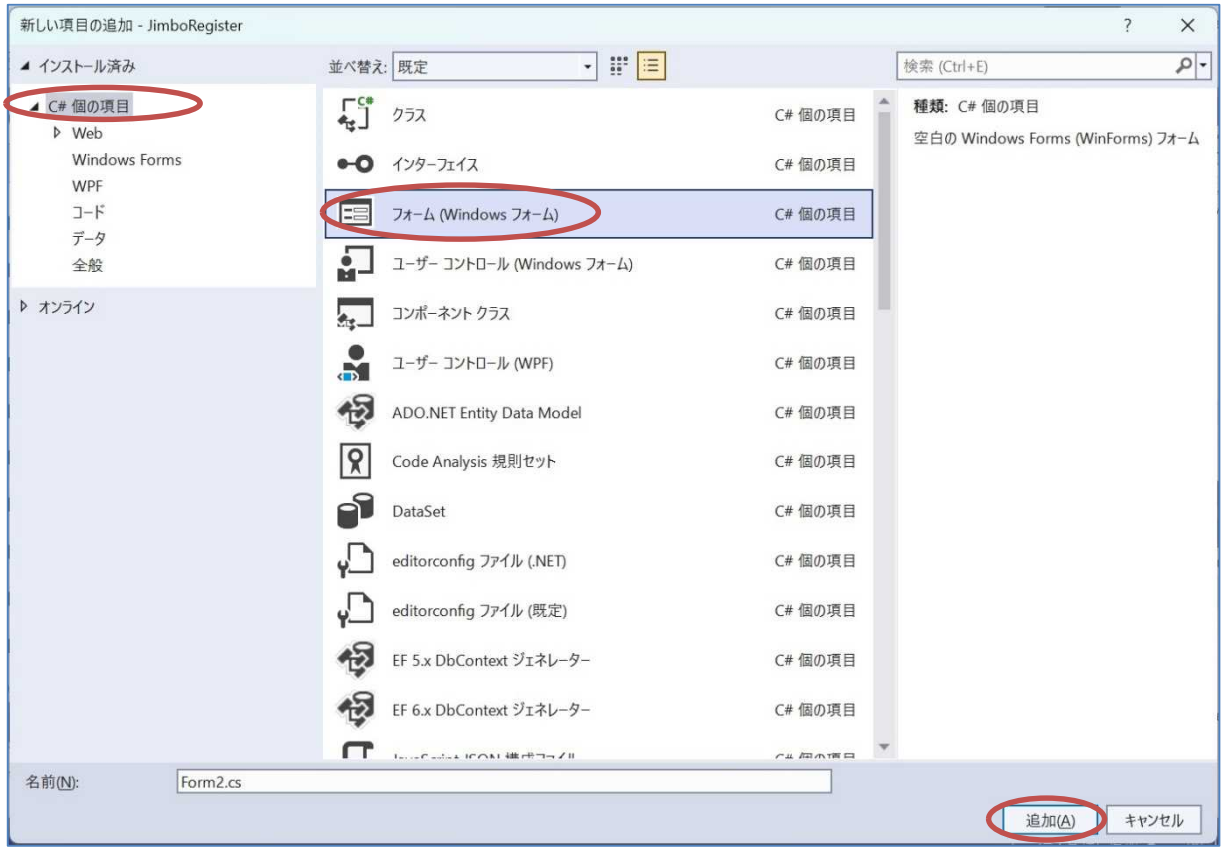
続いて，フォームデザイナー上で button13～button15 をダブルクリックして，イベントハンドラ button13_Click～button15_Click の処理内容を次の様に記述する。

```
        1 個の参照
        private void button13_Click(object sender, EventArgs e)
        {
            if (upc[2] > downc[2])
            {
                num[2]--;
                labelDisp(prod[countud], countud);
                setValues(countud);
                downc[2]++;
            }
        }

        1 個の参照
        private void button14_Click(object sender, EventArgs e)
        {
            if (upc[3] > downc[3])
            {
                num[3]--;
                labelDisp(prod[countud], countud);
                setValues(countud);
                downc[3]++;
            }
        }

        1 個の参照
        private void button15_Click(object sender, EventArgs e)
        {
            if (upc[4] > downc[4])
            {
                num[4]--;
                labelDisp(prod[countud], countud);
                setValues(countud);
                downc[4]++;
            }
        }
```

2) フォームデザイン（Form2）

[プロジェクト] → [Windows フォームの追加]を選択する。

『C# 個の項目』及び『Windows フォーム』が選択されていることを確認して,『追加』を押す。名前は『Form2.cs』のままとする。



追加された Form2 上にラベルを 6 個，テキストボックスを 1 個，ボタンを 1 個貼り付ける。それぞれのプロパティは次の様に設定する。

【Form2 のプロパティ】　Font：14pt

Size：(500, 400)　← (X, Y)　　＊括弧は入力しない。

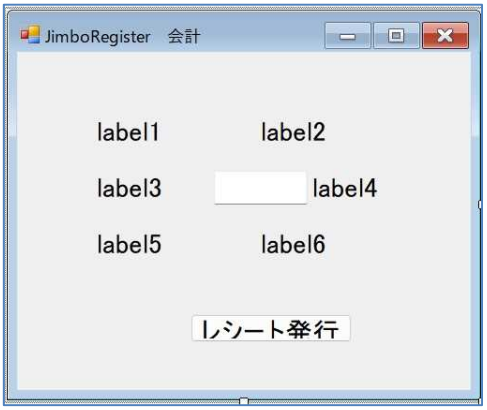
Text：　"JimboRegister　会計"　← 文字列リテラルの中身を入力　　＊二重引用符は入力しない。

【ラベルのプロパティ】　Font：14pt

label1		label2		label3	
Location	(80, 70)	Location	(255, 70)	Location	(80, 130)
label4		label5		label6	
Location	(310, 130)	Location	(80, 190)	Location	(255, 190)

【テキストボックスのプロパティ】　Font：14pt

Location：(210, 127)　← (X, Y)　　＊括弧は入力しない。



【ボタンのプロパティ】　Font：14pt

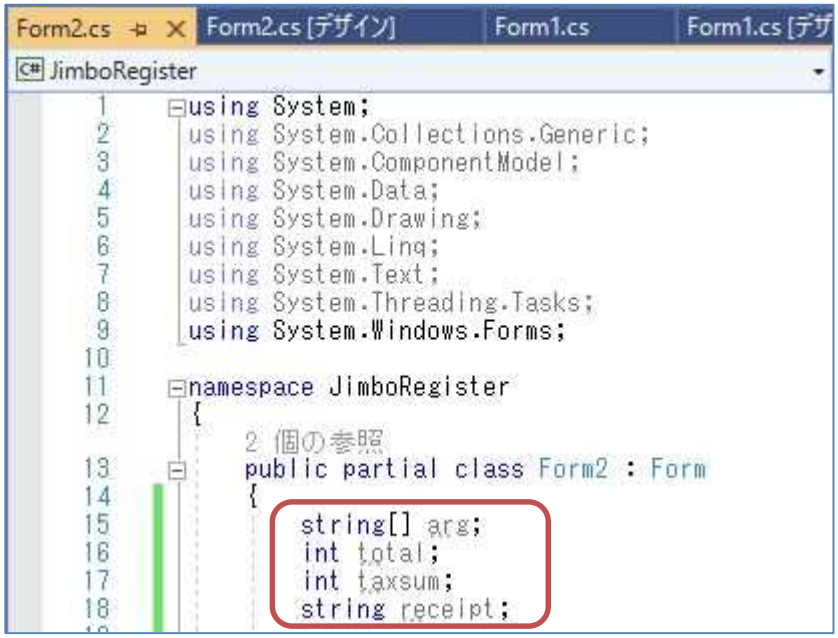
Size：(122, 30)

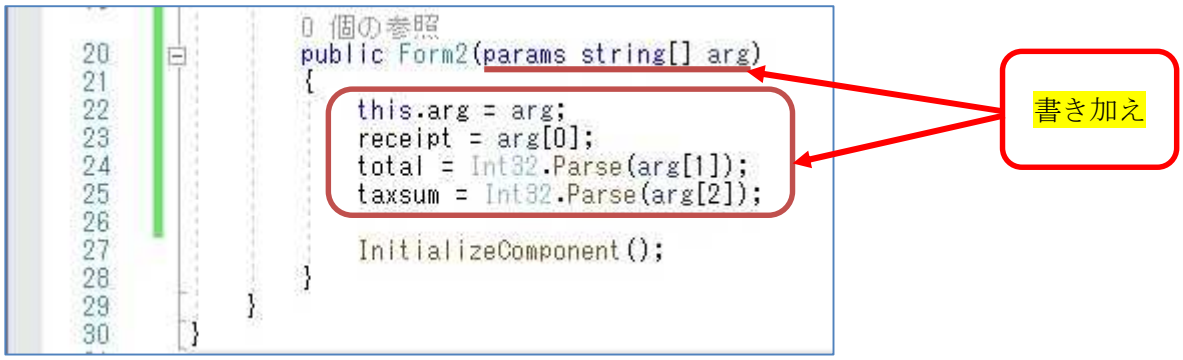
Location：(185, 280)

Text：　"レシート発行"

3) クラスの作成

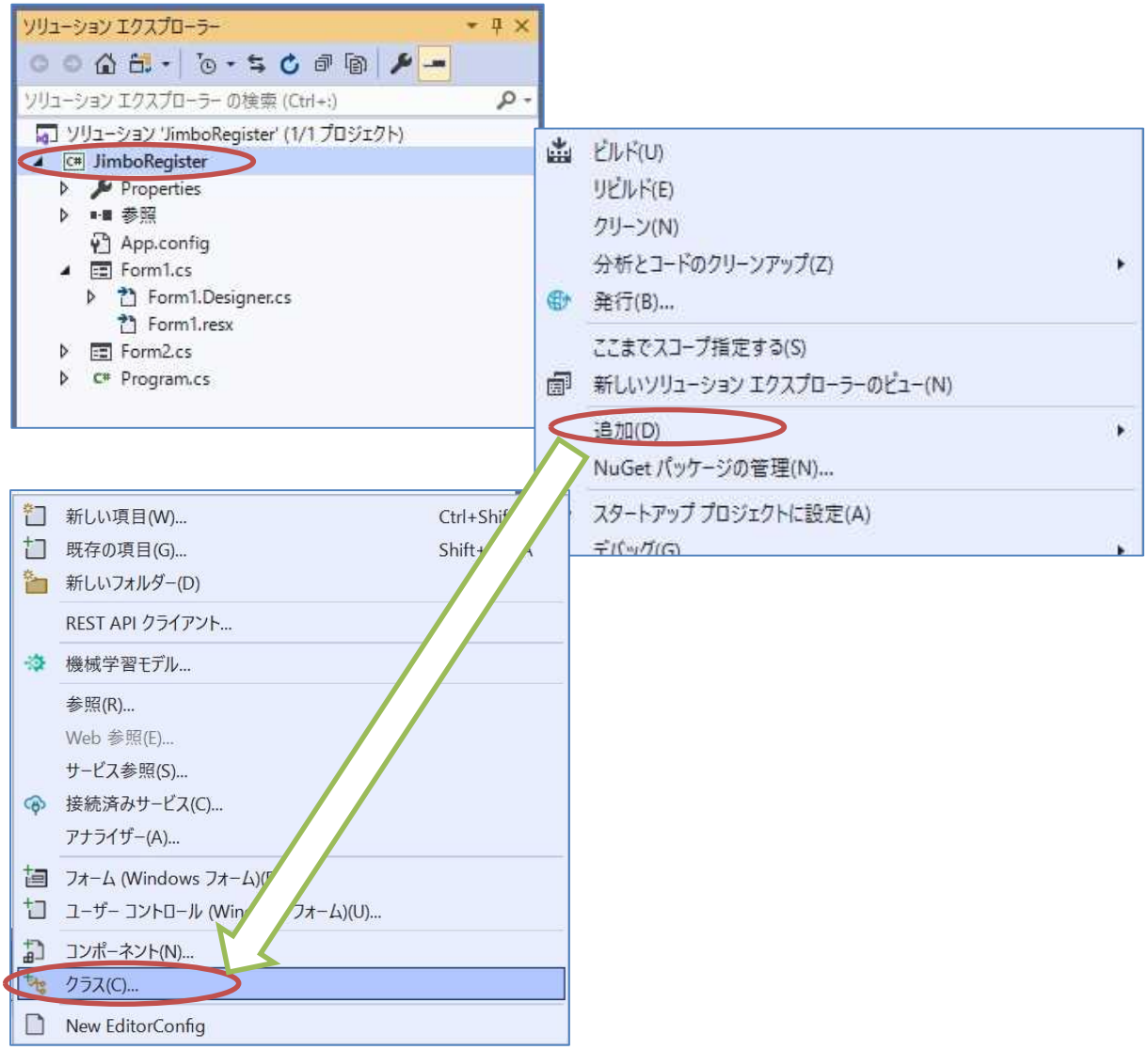
最初に，準備として，Form2 のコーディングの一部を実施しておく。[表示] → [コード]と選択してコードエディタを開き，Form2 クラスのフィールドに配列 arg，変数 total，taxsum，receipt を宣言する。ここで，配列 arg は From1 から受け渡された文字列を格納するために用いる。また，Form2 のコンストラクタに引数を設定し，Form2 のフィールド変数に値を代入する。

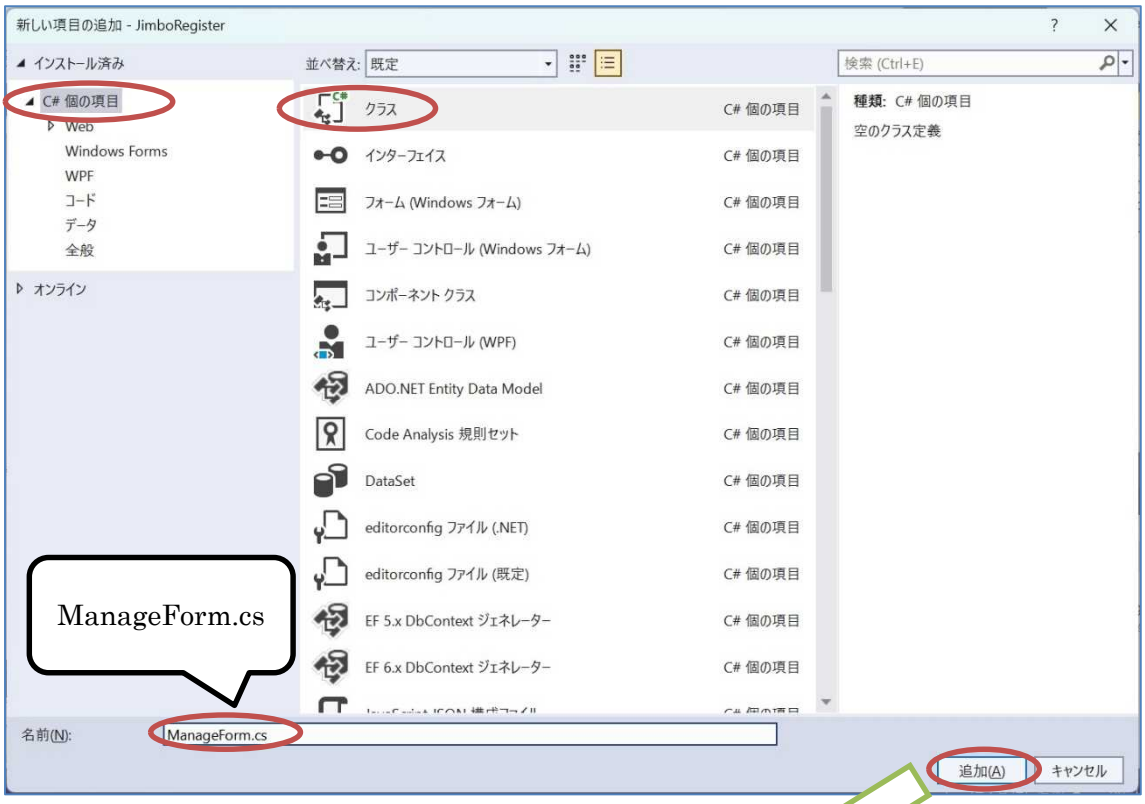


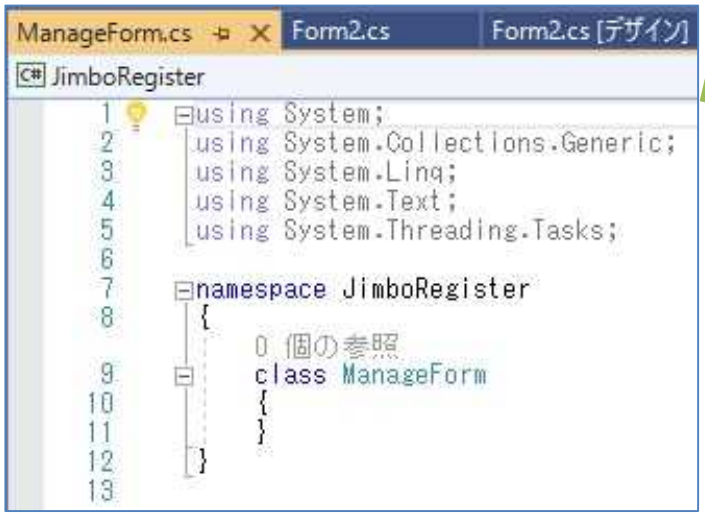


ここで，引数中の params 修飾子は要素数が可変の配列を受け取れることを表す。また，Int32.Parse メソッドは，数値の文字列形式を，それと等価な 32 ビット符号付き整数に変換する。

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，「ManageForm.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 ManageForm クラスにコンストラクタを記述する。

ここで，ShowDialog メソッドはフォームをモーダルダイアログボックス（ダイアログボックスを閉じるまでは，同じアプリケーションの他のウィンドウに対する操作ができないダイアログボックス）として表示する。モーダルダイアログボックスを表示した場合には，閉じる際にインスタンスのリソースを解放する Dispose メソッドが必要となる。

4）コーディング（Form1 及び Form2）

先ず，Form1 で，イベントハンドラ button33_Click の処理内容の記述を次の様に変更する。

```
        1 個の参照
189     private void button33_Click(object sender, EventArgs e)
190     {
191         if (b33flag != 0)
192         {
193             receipt = "レシート¥n";
194         }
195
196         for (int i = 0; i < MAXNUM; i++)
197         {
198             receipt += subrec[i];
199         }
200
201         ManageForm mf = new ManageForm(receipt, total.ToString(), taxsum.ToString());
202
203     //  MessageBox.Show(receipt + "合計 " + String.Format("{0:#,0} 円¥n", total)
204     //      + " (税額 " + String.Format("{0:#,0} 円", taxsum) + ") ");
205
206         b33flag = 1;
207     }
```

ここで，mf は ManageForm クラスのインスタンス変数で，右辺の new ManageForm(receipt, total.ToString(), taxsum.ToString()) は，ManageForm クラスのインスタンスを生成する際に，3 個の文字列をコンストラクタに引き渡して初期化する。

＊ //でコメントアウトした 2 行は削除しても良いが，Form2 の button1_Click で利用するので残してある。

次に，Form2 のフォームデザイナー上でフォームをダブルクリックして，イベントハンドラ Form2_Load の処理内容を次の様に記述する。

```
        1 個の参照
30      private void Form2_Load(object sender, EventArgs e)
31      {
32          label1.Text = "合計金額";
33          label2.Text = String.Format("{0:#,0} 円", total);
34          label3.Text = "お預かり金";
35          label4.Text = "円";
36          label5.Text = "お釣り";
37          label6.Text = "";
38      }
```

続けて，Form2 のフォームデザイナー上でテキストボックスをダブルクリックして，イベントハンドラ textBox1_TextChanged の処理内容を次の様に記述する。

（図は次のページ）

```
1 個の参照
private void textBox1_TextChanged(object sender, EventArgs e)
{
    int change;

    change = Int32.Parse(textBox1.Text) - total;

    if (change >= 0)
    {
        label6.Text = String.Format("{0:#,0} 円", change);
    }
}
```

ここで，change はお釣りを格納する変数である。

最後に，Form2 のフォームデザイナー上で button1 をダブルクリックして，イベントハンドラ button1_Click の処理内容を次の様に記述する。

```
1 個の参照
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show(receipt + "合計 " + String.Format("{0:#,0} 円¥n", total)
        + " (税額 " + String.Format("{0:#,0} 円", taxsum) + ") ");
}
```

この処理は，これまで Form1 の button33_Click に記述されていたものをこちらに移設したものである。

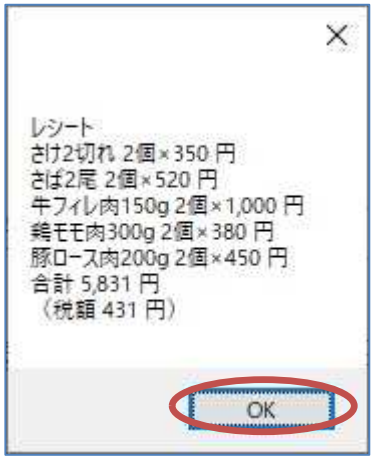
5) 実行

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 5 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。

最後に，『レシート発行』ボタンをクリックする。



提出物：

1）フォームのデザインファイル Form1.Designer.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 Form1.cs をメールに添付して提出する。

3）フォームのデザインファイル Form2.Designer.cs をメールに添付して提出する。

4）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。

5) ManageForm クラスにコンストラクタを記述したソースファイル ManageForm.cs をメールに添付して提出する。

6）実行して，アプリが起動後，5 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 9 回実行結果.jpg（.png も可）をメールに添付して提出する。

7）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 9 回会計.jpg（.png も可）をメールに添付して提出する。

8）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 9 回レシート.jpg（.png も可）をメールに添付して提出する。

9）質問を記述したファイル Prog2_Questions_9th.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。