2023 年 7 月 6 日（木）実施

例外処理

C#言語では，作成したプログラムを実行する際に，記述した処理が想定しない事態によって実行できなくなる場合を**例外**と呼び，その例外への対処，即ち**例外処理**が求められる。

例外処理を行うための try 文の一般形は次のようになる。

```
try
{
   例外を発生させる可能性のある処理
}
catch(例外のクラス名 1 変数 1 )
{
        例外に対処する処理 1
}
catch(例外のクラス名 2 変数 2 )
{
        例外に対処する処理 2
}
          ・・・
finally
{
        try 文の最後に必ず実行される処理
}
```

ここで，catch ブロックと finally ブロックとは何れかを必ず記述する。catch ブロックは必要に応じて，例外を場合分けして複数記述することが出来る。finally ブロックには例外発生時に中断した処理の後処理等を記述する。なお，例外クラスの最も基本的なクラスは System.Exception なので，catch ブロックの引数のクラス名に Exception を記述した場合には，全ての例外クラスに対応することが出来る。

ファイル操作

ファイル(File)とは，データの集合体のことで，JIS(日本工業規格)では，ファイルはレコードの集合体，レコードはデータの集合体と定義されている。ファイル操作は，次の順序で行う。なお，**ストリーム**とは，入力元または出力先を持つ，順序付けられたデータ列である。

1) ストリームを開く
2) ストリームからの入力，ストリームへの出力
3) ストリームを閉じる

C#言語では，システムライブラリの System.IO 名前空間にファイル操作に関係するクラスが複数用意されている。テキストファイルから文字を読み込むためには StreamReader クラスを用い，テキストファイルに文字を書き込むためには StreamWriter クラスを用いる。

ファイル操作を行う場合には，ファイルが開けない例外，開いたファイルからデータが読み込めない例外といった，様々な例外があり得るので，例外処理は不可欠である。そこで，テキストファイルから文字を読み込む場合には，次の様なコードが必要となる（using System.IO; が前提）。

```
StreamReader sr = null;

try
{
    sr = new StreamReader("パス名¥¥ファイル名")
    文字データの読み込み処理
    読み込んだデータを用いた処理
}
catch (Exception ex)
{
    例外を知らせるメッセージの表示
}
finally
{
    if (sr != null)
    {
        sr.Close;
    }
}
```

この使い方の using は using ディレクティブ

＊using ディレクティブを用いない場合は System.IO.StreamReader

これに対して，using ステートメント（using ディレクティブとは別物）を用いると，オブジェクトの終了処理を行う Dispose メソッドを呼び出す。using ブロックで例外が発生した場合でも必ず Dispose メソッドが呼び出されることから，上のコードの様な finally ブロックは不要となり，次の様に書き換えることが出来る。

```
try
{
    using (StreamReader sr = new StreamReader("パス名¥¥ファイル名"))
    {
        文字データの読み込み処理
        読み込んだデータを用いた処理
    }
}
catch (Exception ex)
{
    例外を知らせるメッセージの表示
}
```

この使い方の using は using ステートメント

＊ ¥¥は**エスケープシーケンス**と呼ばれ，文字列リテラルにそのままでは書き込めない文字（二重引用符，改行記号等）をエスケープ文字¥に続けて書くことで書き込み可能としている。

また，同様に，テキストファイルに文字を書き込む場合には，次の様なコードが必要となる（using System.IO; が前提）。

```
try
{
    using (StreamWriter sw = new StreamWriter("パス名¥¥ファイル名"))
    {
        文字データの書き込み処理
    }
}
catch (Exception ex)
{
    例外を知らせるメッセージの表示
}
```

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，例外処理の基本及びファイル処理について学ぶ。

手順

1）コーディング（前回の続き・発展）

Form1.cs で，イベントハンドラ button21_Click～button28_Click の処理内容のうち，2 行目～10 行目までを，新規にメソッド TreatProd を作成して記述する。第 1 引数には商品名，第 2 引数には商品ボタンの番号(button21 が 0 番)で引き渡される。また，button21_Click では TreatProd を呼び出す様に書き直す。IndexOutOfRangeException は境界外のインデックスを使用して配列またはコレクションの要素にアクセスしようとしたときの例外に対応するクラスである。

```
        1 個の参照
146     private void TreatProd(string s, int n)
147     {
148         try
149         {
150             labelSet(lproduct[count], s);
151             buttonVis(upb[count], 1);
152             buttonVis(downb[count], 1);
153             pretotal = total;
154             pretaxsum = taxsum;
155             prod[count] = n;
156             labelDisp(n, count);
157             setValues(count);
158             count++;
159         }
160         catch(IndexOutOfRangeException e)
161         {
162             MessageBox.Show(e +
163                 "¥n 現在の仕様では登録出来る商品は10種類までです。");
164         }
165     }
```

イベントハンドラ button21_Click は商品名と商品ボタンの番号とを実引数として TreatProd を呼び出す様に書き換える。

```
        1 個の参照
136     private void button21_Click(object sender, EventArgs e)
137     {
138         if (pflag[0] == 0)
139         {
140             sprod = button21.Text;
141             TreatProd(sprod, 0);
142             pflag[0] = 1;
143         }
144     }
```

button22_Click～button28_Click の処理内容も同様に書き換える。

```
        1 個の参照
184     private void button22_Click(object sender, EventArgs e)
185     {
186         if (pflag[1] == 0)
187         {
188             sprod = button22.Text;
189             TreatProd(sprod, 1);
190             pflag[1] = 1;
191         }
192     }
```

```
        1 個の参照
264     private void button23_Click(object sender, EventArgs e)
265     {
266         if (pflag[2] == 0)
267         {
268             sprod = button23.Text;
269             TreatProd(sprod, 2);
270             pflag[2] = 1;
271         }
272     }
```

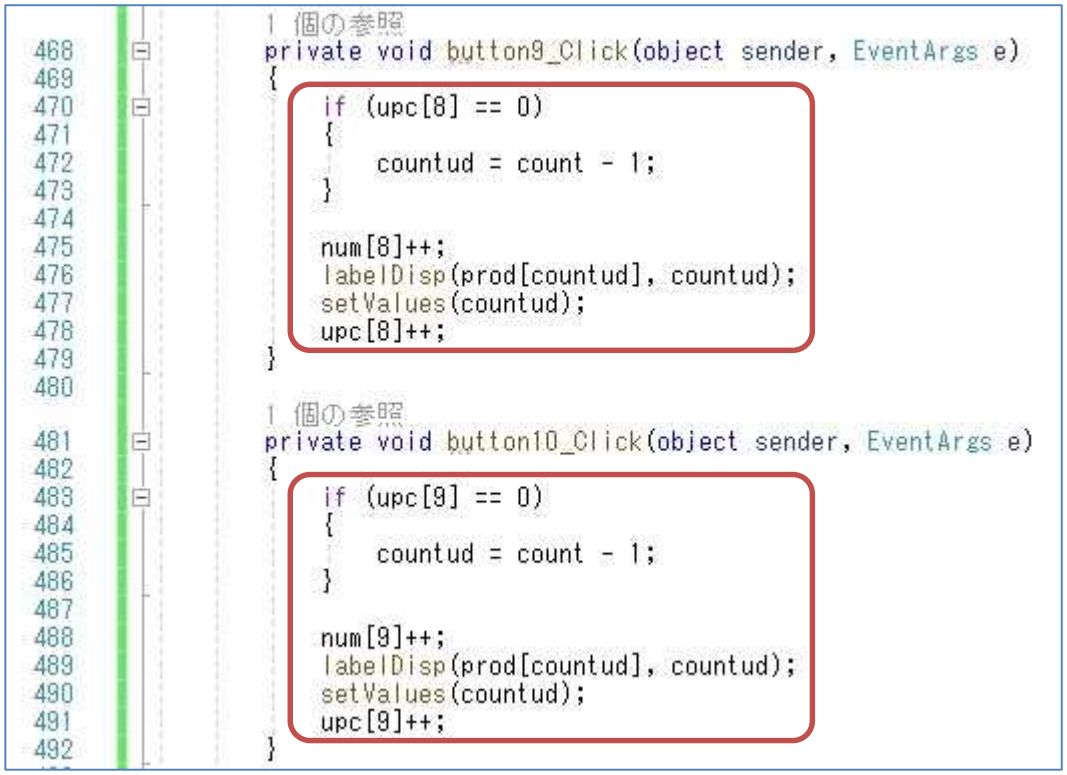
・・・

```
        1 個の参照
386     private void button28_Click(object sender, EventArgs e)
387     {
388         if (pflag[7] == 0)
389         {
390             sprod = button28.Text;
391             TreatProd(sprod, 7);
392             pflag[7] = 1;
393         }
394     }
```

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 button9 及び button10 をダブルクリックして，イベントハンドラ button9_Click 及び button10_Click の処理内容を次の様に記述する。

次に，フォームデザイナー上で button19 及び button20 をダブルクリックして，イベントハンドラ button19_Click 及び button20_Click の処理内容を次の様に記述する。

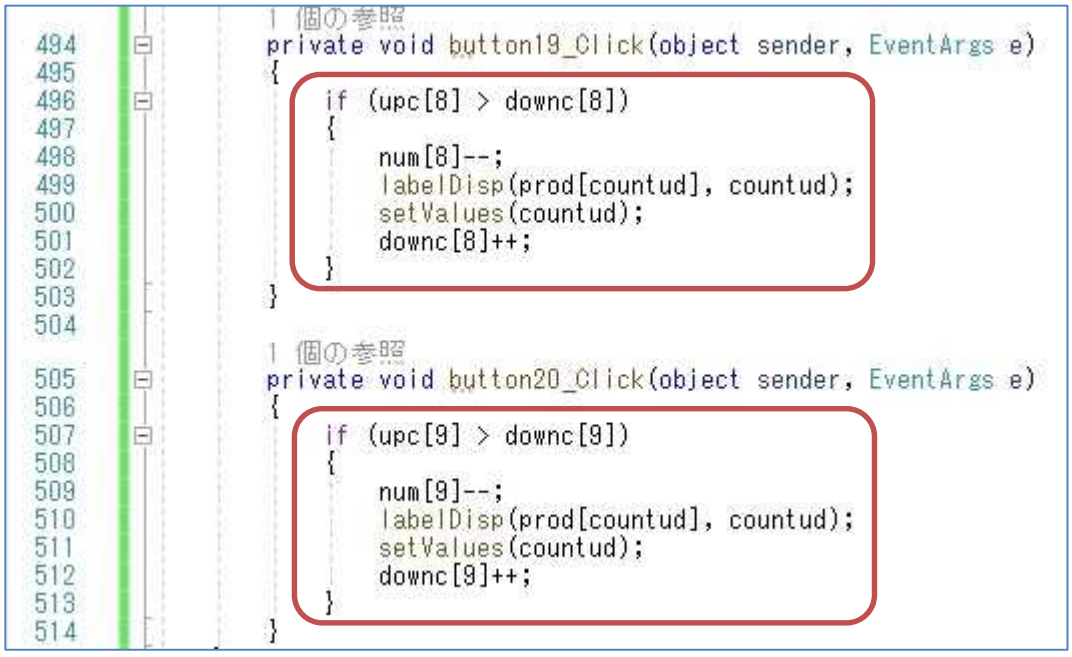


```
1 個の参照
private void button9_Click(object sender, EventArgs e)
{
    if (upc[8] == 0)
    {
        countud = count - 1;
    }

    num[8]++;
    labelDisp(prod[countud], countud);
    setValues(countud);
    upc[8]++;
}

1 個の参照
private void button10_Click(object sender, EventArgs e)
{
    if (upc[9] == 0)
    {
        countud = count - 1;
    }

    num[9]++;
    labelDisp(prod[countud], countud);
    setValues(countud);
    upc[9]++;
}
```



```
1 個の参照
private void button19_Click(object sender, EventArgs e)
{
    if (upc[8] > downc[8])
    {
        num[8]--;
        labelDisp(prod[countud], countud);
        setValues(countud);
        downc[8]++;
    }
}

1 個の参照
private void button20_Click(object sender, EventArgs e)
{
    if (upc[9] > downc[9])
    {
        num[9]--;
        labelDisp(prod[countud], countud);
        setValues(countud);
        downc[9]++;
    }
}
```

2) フォームへのコントロールの追加及びコーディング

Form1 にラベルを 1 個，ボタンを 2 個追加する。それぞれのプロパティは次の通りである。

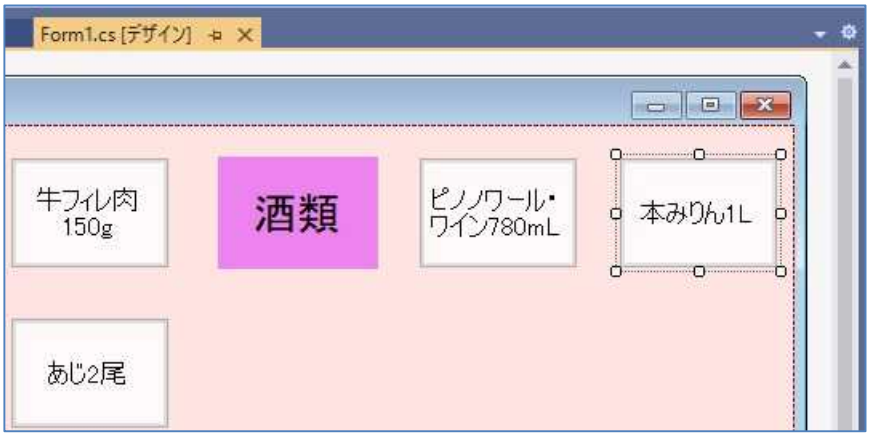
【商品カテゴリラベル label36 のプロパティ】

AutoSize：false　Size：(100, 70)　TextAlign：MiddleCenter　Font：20pt

BackColor：Violet　Location：(540, 20)　Text：″酒類″

【商品ボタンのプロパティ】BackColor：Snow　Size：(100，70)

button34		button35	
Location	(665，20)	Location	(790，20)
Text	″ピノノワール・ワイン 780mL″	Text	″本みりん 1L″



続けて，Form1 クラスのフィールドを書き換える。

```
26        const int MAXNUM = 10;
27        const int MAXPROD = 14;
28        int[] prod = new int[MAXNUM];
29        int[] num = new int[MAXNUM];
30        int[] upc = new int[MAXNUM];
31        int[] downc = new int[MAXNUM];
32        int[] subtotal = new int[MAXNUM];
33        int[] subtax = new int[MAXNUM];
34        string[] subrec = new string[MAXNUM];
35        int[] pflag = new int[MAXPROD];
36
37        readonly int[] price = new int[] { 450, 380, 1000,
38            520, 350, 390, 290, 150, 198, 250, 230, 210, 3200, 1640 };
39        readonly int[] tax = new int[] { 8, 8, 8, 8, 8, 8,
40            8, 8, 8, 8, 8, 8, 10, 10 };
41        readonly string[] product = new string[]
```

Form1 のフォームデザイナー上で button29～button32，button34 及び button35 をダブルクリックして，イベントハンドラ button29_Click～button32_Click，button34_Click 及び button35_Click の処理内容を次の様に記述する。

```
516       1 個の参照
          private void button29_Click(object sender, EventArgs e)
517       {
518           if (pflag[8] == 0)
519           {
520               sprod = button29.Text;
521               TreatProd(sprod, 8);
522               pflag[8] = 1;
523           }
524       }
```

```
526         1 個の参照
            private void button30_Click(object sender, EventArgs e)
527         {
528             if (pflag[9] == 0)
529             {
530                 sprod = button30.Text;
531                 TreatProd(sprod, 9);
532                 pflag[9] = 1;
533             }
534         }
```

```
536         1 個の参照
            private void button31_Click(object sender, EventArgs e)
537         {
538             if (pflag[10] == 0)
539             {
540                 sprod = button31.Text;
541                 TreatProd(sprod, 10);
542                 pflag[10] = 1;
543             }
544         }
```

```
546         1 個の参照
            private void button32_Click(object sender, EventArgs e)
547         {
548             if (pflag[11] == 0)
549             {
550                 sprod = button32.Text;
551                 TreatProd(sprod, 11);
552                 pflag[11] = 1;
553             }
554         }
```

```
556         1 個の参照
            private void button34_Click(object sender, EventArgs e)
557         {
558             if (pflag[12] == 0)
559             {
560                 sprod = button34.Text;
561                 TreatProd(sprod, 12);
562                 pflag[12] = 1;
563             }
564         }
```

```
566         1 個の参照
            private void button35_Click(object sender, EventArgs e)
567         {
568             if (pflag[13] == 0)
569             {
570                 sprod = button35.Text;
571                 TreatProd(sprod, 13);
572                 pflag[13] = 1;
573             }
574         }
```

続けて，DispReceipt2 クラスのフィールドを書き換える。

```
10          3 個の参照
            class DispReceipt2 : DispReceipt
11          {
12              int cash;
13              int change;
14              public string result2;
15
```

result2 への外部からのアクセスを可能とする。

Form2 クラスのイベントハンドラ button1_Click の内容を変更して，DispReceipt2 クラスのインスタンスの result2 を利用し，テキストファイルへのレシート内容の書き出しを行える様にする。そのためには，先ず，using ディレクティブとして using System.IO; を加える。

```
1   using System;
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.IO;
7   using System.Linq;
8   using System.Text;
9   using System.Threading.Tasks;
10  using System.Windows.Forms;
11
12  namespace JimboRegister
13  {
        4 個の参照
14      public partial class Form2 : Form
15      {
```

```
        1 個の参照
54      private void button1_Click(object sender, EventArgs e)
55      {
56          string mstr = "";
57
58          DispReceipt2 ds2 = new DispReceipt2(receipt, total.ToString(), taxsum.ToString(),
59              cash.ToString(), change.ToString());
60          ds2.DispMB();
61
62          try
63          {
64              using (StreamWriter sw = new StreamWriter("Receipt.txt"))
65              {
66                  sw.WriteLine(ds2.result2);
67              }
68
69              mstr = "ファイルに書き出します。";
70          }
71          catch (Exception ex)
72          {
73              mstr = ex.Message;
74          }
75          finally
76          {
77              MessageBox.Show(mstr);
78          }
79      }
```

3) 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クリック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 10 件分行う。

次に，11 件目の商品ボタンをクリックしたらエラーメッセージが表示されることを確かめ，OK ボタンをクリックする。

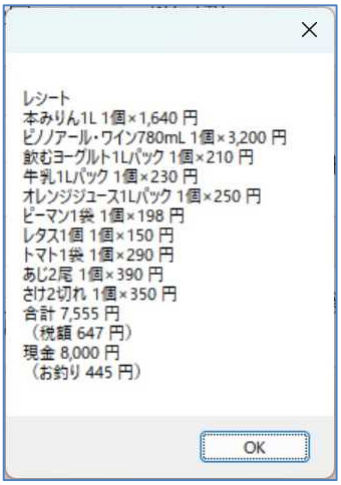
更に，『お預かり金』の金額をテキストボックスに入力し，『会計』ボタンをクリックする。

最後に，メッセージボックスの OK ボタンをクリックした後に新たなメッセージボックスが表示されることを確かめる。


JimboRegister
本みりん1L
1 ×1,640 円
ピノノワール・ワイン780mL
1 ×3,200 円
飲むヨーグルト1Lパック
1 ×210 円
牛乳1Lパック
1 ×230 円
オレンジジュース1Lパック
1 ×250 円
ピーマン1袋
1 ×198 円
レタス1個
1 ×150 円
トマト1袋
1 ×290 円
あじ2尾
1 ×390 円
さけ2切れ
1 ×350 円
合計 7,555 円
肉類
豚ロース肉 200g
鶏モモ肉 300g
牛フィレ肉 150g
酒類
ピノノワール・ワイン780mL
本みりん1L
魚貝類
さば2尾
さけ2切れ
あじ2尾
野菜
トマト1袋
レタス1個
ピーマン1袋
飲料
オレンジジュース1Lパック
牛乳1Lパック
飲むヨーグルト1Lパック
会計


JimboRegister
本みりん1L
1 ×1,640 円
ピノノワール・ワイン780mL
1 ×3,200 円
飲むヨーグルト1Lパック
1 ×210 円
牛乳1Lパック
1 ×230 円
オレンジジュース1Lパック
1 ×250 円
ピーマン1袋
1 ×198 円
レタス1個
1 ×150 円
トマト1袋
1 ×290 円
あじ2尾
1 ×390 円
さけ2切れ
1 ×350 円
合計 7,555 円
肉類
豚ロース肉 200g
鶏モモ肉 300g
牛フィレ肉 150g
酒類
ピノノワール・ワイン780mL
本みりん1L
魚貝類
さば2尾
さけ2切れ
あじ2尾
野菜
トマト1袋
レタス1個
ピーマン1袋
飲料
オレンジジュース1Lパック
System.IndexOutOfRangeException: インデックスが配列の境界外です。
場所 JimboRegister.Form1.TreatProd(String s, Int32 n) 場所 C:¥Users¥jimbo¥source¥repos¥JimboRegister¥JimboRegister¥Form1.cs:行 150
現在の仕様では登録出来る商品は10種類までです。
OK
会計


JimboRegister 会計
合計金額
7,555 円
お預かり金
8000
円
お釣り
445 円
レシート発行


レシート
本みりん1L 1個×1,640 円
ピノアール・ワイン780mL 1個×3,200 円
飲むヨーグルト1Lパック 1個×210 円
牛乳1Lパック 1個×230 円
オレンジジュース1Lパック 1個×250 円
ピーマン1袋 1個×198 円
レタス1個 1個×150 円
トマト1袋 1個×290 円
あじ2尾 1個×390 円
さけ2切れ 1個×350 円
合計 7,555 円
（税額 647 円）
現金 8,000 円
（お釣り 445 円）
OK

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）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。
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，10 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 11 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，11 件目の商品ボタンをクリックして表示されたエラーメッセージのスクリーンショット第 11 回例外処理.jpg（.png も可）をメールに添付して提出する。
7）『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後に表示された「ファイルに書き出します。」と書かれたメッセージボックスのスクリーンショット第 11 回メッセージ.jpg（.png も可）をメールに添付して提出する。
8）C ドライブの［ユーザー］→［ユーザー名］→［source］→［repos］→［JimboRegister］→［JimboRegister］→［bin］→［Debug］と辿った先のフォルダに作成されたテキストファイル Receipt.txt
9）質問を記述したファイル Prog2_Questions_11th.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。