**<C#復習>**

**○Visual Studioの使い方**

・上にあるファイル名が書いているところの緑色の矢印で実行される

・課題の提出の前にソリューションファイルを開き、右のソリューションタブで右クリックして「ソリューションのクリーンをする」

**○テンプレの形**

・基本の形

using System;

class クラス名{

private 型 変数名;

public static クラス名 Parse(string s){

int i = Int32.Parse(s);

return new クラス名;

}

または

public クラス名(int c0){

変数名 = c0;

}

public クラス名(型 引数名){ // コンストラクタ宣言

変数名 = 引数名;

}

public 型 メソッド1(){}

public 型 メソッド2(){}

}

class Program{

static void Main(string[] args){

本文

}

}

**○基本**

・最初に「using System;」とかく

・クラス宣言 class クラス名{ }

・フィールド宣言：クラスの中で用いる共通の名前。「変数名.フィールド名」として読み書きをすることができる。

「修飾子 型 フィールド名」または「修飾子 型 フィールド名 = 初期値」

例. private int counter;

使うときは「変数名.counter」とする

・**コンストラクタ宣言**：構造体のようなもの、必ずクラスの中でやるようにする

修飾子 クラス名(型 名前, 型 名前, … ) { 文 }

例. public Counter(int c0) { count = c0; }

・コンストラクタ呼び出し（インスタンス名を設定する、クラスに紐づける名前）

クラス名 インスタンス名 = 値;

例. Counter c = new Counter(0);

・**メソッド宣言**：自作の関数のこと

修飾子 返り値の型 メソッド名(型 名前, … ){ 文 }

返り値がないときはvoidを型名とする。

例. public int GetCount(){ return count; }

例. public void Reset(){ count = 0; }

例. static void Main(string[] args){ Console.WriteLine(“Hello World!”); }

・メソッド呼び出し

コンストラクタ名.メソッド名();

例. c.Inc();

・staticメソッド：インスタンスではなくクラスに属するメソッド

例

public static Counter Parse(string s);{

int i = Int32.Parse(s);

return new Counter(i);

}

・型変換

int → string : string s = i.ToString();

string → int : int i = Int32.Parse(s);

**〇クラスの継承**

・元のクラスAのメソッドをクラスBに受け継がせることを継承という。

・下のような形になる

class 元クラス{

public virtual 型 メソッド名(引数){ 文 } // 仮想メソッド

}

class 新クラス : 元クラス{

protected 型 変数名;

public 新クラス(型 変数名, … ) : base(元のクラスで使ってた変数名){

変数名 = 変数名;

}

public override 型 メソッド名(引数){

base.メソッド名(引数);

追加したい文

}

}

・使うときは「新クラス名 変数名 = new 新クラス名(引数)」とする

・オーバーライド：元クラスの編集ができる

元クラスでのちに編集する予定のメソッドには「public virtual 型 メソッド名(引数){}」とする

新クラスで編集するときは「public override 型 同じメソッド名(引数){}」とする。元クラスを呼ぶときは「base.メソッド名(引数)」とする。別に必ずやる必要はない。

**○配列**

・配列の宣言1 型[] 名前 = new 型[要素数];

string [] names = new string[3];

・配列の宣言2 型[] 名前 = {値1, 値2, 値3};

int[] nums = new int[] {1, 10, 100};

・配列の代入 名前[インデックス] = 値;

names[0] = “yuton”;

names[1] = “hazu”;

names[2] = “nakano“;

・配列の要素数 名前.Length

names.Length; // 3

・配列の要素を全て取り出すにはforeach文が便利。foreach(型 名前 in 配列){}

foreach(var name in names){ Console.WriteLine(name); }

・多次元配列の宣言 int[,] v = {{11, 12, 13}, {21, 22, 23}};

・多次元配列の要素数 v.Length // 6

・多次元配列の次元数 v.Rank // 2

・多次元配列のi次元の要素数 v.GetLength(0) // 2

v.GetLength(1) // 3

**○配列の進化版 List<T>**

・配列の定義をしてから追加、削除を行うことができる

・使うには「using System.Collections.Generic;」を文頭につける

・宣言 List<型> 名前 = new List<型>();

List<string> names = new List<string>();

・値の追加 名前.Add(値);

names.Add(“apple”);

names.Add(“orange”);

names.Add(“banana”);

・値の変更

names[1] = “mikan”;

・値の削除 名前.Remove(値);

names.Remove(“mikan”);

・指定したインデックスの要素を削除 名前.RemoveAt(インデックス);

names.RemoveAt(0); // “apple”が消える

・指定したインデックスに要素をねじ込む 名前.Insert(インデックス, 値);

names.Insert(0, “melon”);

・配列の要素数 名前.Count

names.Count // 2

**○タプル**

・タプルの宣言　(型1, 型2, 型3) 名前 = (値1, 値2, 値3);

(string, int, int) a = (“apple“, 120, 3);

・タプルの値の取り出し 名前.Itemインデックス

a.Item1 // “apple”

a.Item2 // 120

a.Item3 // 3

**○辞書 Dictionary**

・KeyとValueを紐付ける

・宣言 Dictionary<Keyの型, Valueの型> 名前 = new Dictionary<Key型, Value型>();

Dictionary<string, int> dict = new Dictionary<string, int>();

その後に{}で囲んでKey, Valueを並べれば初期値も一緒に設定できる

Dictionary<string, int> dict1 = new Dictionary<string, int>(){

{Key1, Value1}, {Key2, Value2}

};

Dictionary<string, int> dict = new Dictionary<string, int>(){

[Key1] = Value1, [Key2] = Value2

};

・要素の追加 名前.Add(Keyの値, Valueの値)

dict.Add(“yuton”, 20);

dict.Add(“kakao”, 49);

dict.Add(“tsumu”, 50);

・Valueの取得 名前[Key]

dict[“yuton“] // 20

・Valueの更新 名前[Key] = 値;

dict[“yuton“] = 21;

・追加&更新 「名前[Key] = Value;」とするとKeyがある場合更新されて、Keyがない場合新しく追加される

・Key, Valueの削除 名前.Remove(Key);

dict.Remove

・指定したKeyが存在するか調べる 名前.ContainsKey(Key)

if(dict.ContainsKey(“tsumu“)) {} // 存在するならtrue

if(dict.ContainsKey(“milk“)) {} // 存在しないならfalse

・foreach文を用いてKeyとValueを同時に取得する

foreach(KeyValuePair<string, int> n in dict){

Console.WriteLine(“{0} : {1}”, n.Key, n.Value);

}

または

foreach(var kv in dict){}

・foreach文を用いてKeyのみ取得

foreach(string k in dict.Keys){}

・foreach文を用いてValueのみ取得

foreach(int v in dict.Values){}

**○プロパティ**

・フィールドと同じように「名前.プロパティ名」で読み書きができる

・プロパティの宣言 public 型 名前

・getで読む（取り出す）ときの挙動、setで書くときの挙動

private int score = 0;

public int Score{

get { return score; } // if文で条件分岐させるたりする

set { score = value; } // valueは書き込むときの値

}

上のように通常のフィールドと同じ挙動をするときは省略できる

public int Score{ get; set;}

・読み専用にしたいならgetのみを書けば良い

**○ファイル入出力**

・最初に「using System.IO;」を書く

・プロジェクト名.csprojの<PropertyGroup>と</PropertyGroup> の間の一番上に以下の行を追加する．

<StartWorkingDirectory>.</StartWorkingDirectory>

【入力】

・ファイルの名前を入れた変数を作ると便利

string file\_name = “./mytext.txt”;

・ファイルの内容を全て1つのstringとして読み取る

string txt = File.ReadAllText(“./ファイル名”);

・ファイルの内容を各行のstring型配列として読み取る

string[] lines = File.ReadAllLines(“./ファイル名”);

foreach(var line in lines){} // 使うときはforeachが便利

【出力】

・1つの文字列の内容を一気に書く File.WriteAllText(“./ファイル名”, 変数);

string[] txts = { “リンゴ”, “オレンジ”, “バナナ”};

string txt = String.Join(“\n”, txts);

File.WriteAllText(“./test1”, txt);

・文字列配列を改行して全て書き込む

File.WriteAllLines(“./test2”, txts);

**○GUIプログラミングの基本**

・最初に「using Eto.Forms;」と「Eto.Drawing;」を書く

・テンプレ

public partial class MainForm : Form{

public MainForm(){ // GUIの部品をここに書く }

}

・string Title : ウィンドウのタイトルを表すプロパティ

Title = “Test”;

・Size ClientSize : 中身のサイズ

ClientSize = new Size(500, 500)

・Content Control : ウィンドウの中身となるGUI部品

Content = mainPart; // mainPartの中にGUI部品を記述する

**〇GUI部品**

【Label】入力できないテキストを画面に表示する

・Labelという型（int, stringと同じようなもの）

・名前を付けて使うときは「Label 変数名 = new Label { Text = “表示テキスト” };」として「Items.Add(変数名);」とする

Label label = new Label { Text = “Memo” };

【Button】クリックできるボタン

・Buttonという型（int, stringと同じようなもの）

・名前を付けて使うときは「Button 変数名 = Button { Text = “ボタンに表示する文”};」として「Items.Add(変数名);」とする

Button countButton = new Button{Text = “Load”};

・クリック時のイベントハンドラの登録「変数名.Click += (s, e) =>{文};」

・クリック時のイベントハンドラの解除「変数名.Click -= (s, e) =>{文};」

・クリック時のイベントハンドラのテンプレ

int cnt = 0;

countButton.Click += (s, e) =>{

cnt ++;

countLabel.Text = cnt.ToString();

}

【TextBox】1行のテキストを入力できる欄

・TextBoxという型（int, stringと同じようなもの）

・名前を付けて使うときは「TextBox 変数名 = new TextBox {PlaceholderText = “文”}」として使うときは「Items.Add(変数名);」とする

【TextArea】複数行のテキストを入力できる欄

・TextAreaという型（int, stringと同じようなもの）

・名前を付けて使うときは

「TextArea 変数名 = new StackLayoutItem(new TextArea {PlaceholderText = “文”}, true);」として使うときは「Items.Add(変数名);」とする。trueにすると縦に伸びる

【レイアウト】コントロールの並べ方

・レイアウトの種類

StackLayout : 横または縦の一方向にコントロールを並べる

TableLayout : 表のセルのようにコントロールを配置する

DynamicLayout : 縦と横を切り替えつつコントロールを並べる

StackLayout s = new StackLayout();

・上では横にコントロールを並べたいが下では縦にコントロールを並べたいときは2つのStackLayoutを使うことができる。しかし、Contentに使うことができるのは1つのみなので片方は片方の中に内包するように表記する

StackLayout yoko = new StackLayout(){ 上に配置するGUI部品 };

StackLayout tate = new StackLayout(){

yoko; //

下に配置するGUI部品

};

Content = tate;

・プロパティの使い方

s.Padding = 10; // 外のスペース

s.Spacing = 5; // 各コントロール間のスペース

s.Orientation = Orientation.Vertical; // コンポーネントの配置方向、横ならHorizontal

s.HorizontalContentAlignment = HorizontalAlignment.Stretch; // 横方向に伸ばす

s.Items.Add(GUI部品); // GUI部品をそれぞれ書く。1つの{}の中にも書ける

【簡単な書き方】

StackLayout 名前 = new StackLayout(){

Padding = 10, Spacing = 5,

Orientation = Orientation.Vertical,

HorizontalContentAlignment = HorizontalAlignment.Stretch,

Items = {

new Label {},

new Button {},

new TextBox {},

変数名 // 外で定義したGUI部品の名前は書くだけでよい

}

};

Content = 名前;

**○お絵かきツール Eto.Forms**

・最初に「using Eto.Forms;」「using Eto.Drawing;」とかく

・DrawableにPaintすることで絵をかく

Drawable drawable = new Drawable();

drawable.Paint += (s, pe) =>{

Graphics g = pe.Graphics;

// g.○○で図形を描画する

}

Content = drawable;

【Colorメソッド】Color.○○ または Colors.○○

・Colors.Red, Colors.Blue, Colors.Black, Colors.Whiteで有名な色

・Color.FromRgb(0xFFFFFF)でカラーコードを指定した色

・Color cの透明度を変えるには4つの引数に分割する必要がある。4つめが透明度を表す。値域は1～256。c.Rとかc.Gとかc.Bで三原色を取り出せる

(赤, 緑, 青, 透明度) = (float, float, float, float)

【Graphiccsメソッド】g.○○とする

・g.Clear(Color); 描画領域の背景

・g.DrawLine(Color, PointF, PointF); 太さ1の線分

・g.DrawLine(new Pen(Color, float), PointF, PointF ); 太さを指定した線分

・g.DrawRectangle(Color, RectangleF); 長方形

・g.FillRectangle(Color, RectangleF); 色の塗られた長方形

g.RectangleF = (new PointF, new SizeF) = ((float, float), (float, float))

・g.DrawEllipse(Color, RectangleF); 楕円

・g.FillEllipse(Color, RectangleF); 色の塗られた楕円

・g.DrawPolygon(new Pen(Color, float), PointFの配列); PointFに囲まれた多角形

・g.DrawText(Font, Color, PointF, sting); 指定したフォントでテキストを描く

Font = (SystemFont, float)

【フォントに関して】

・フォントファミリーと文字サイズを指定して文字列を描画することができる

// フォントファミリーは元からあるリストを活用する

DropDown fontfamilyChooser = **new** DropDown { };

fontfamilyChooser.DataStore = Fonts.AvailableFontFamilies;

fontfamilyChooser.ItemTextBinding = Binding.Delegate((object o) => o.ToString());

// 文字サイズはNumericStepperを用いる

NumericStepper sizeSelector = **new** NumericStepper{

MinValue = 5,

MaxValue = 150,

Increment = 1,

Value = 25

};

// 入力するテキストボックス

TextBox textToDraw = **new** TextBox { PlaceholderText = "Text to Draw" };

Drawable d = **new** Drawable();

// 実際に描画するところ

d.Paint += (s, pe) => {

pe.Graphics.Clear(Colors.White);

FontFamily ff = (FontFamily)fontfamilyChooser.SelectedValue;

**if** (ff != **null**){

pe.Graphics.DrawText(**new** Font(ff, (int)sizeSelector.Value), Colors.Black, **new** PointF(0, 0), textToDraw.Text);

}

};

// d.Invalidate()で描画の更新をする

fontfamilyChooser.SelectedIndexChanged += (s, e) => d.Invalidate();

sizeSelector.ValueChanged += (s, e) => d.Invalidate();

textToDraw.TextChanged += (s, e) => d.Invalidate();

// レイアウト

Content = **new** StackLayout{

Padding = 10,

Spacing = 5,

HorizontalContentAlignment = HorizontalAlignment.Stretch,

Items ={

**new** StackLayout{

Orientation = Orientation.Horizontal,

Items = { fontfamilyChooser, sizeSelector }

},

textToDraw,

**new** StackLayoutItem (d, **true**)

}

};

【マウスイベント】

・マウスのクリックに反応して関数を呼び出せる

・Drawable dのd.○○として使う

・d.MouseDown : マウスのクリック

・d.MouseUp : マウスのクリックを離す

・d.MouseMove : マウスのドラッグ（クリックしながらの動作）

・d.MouseWheel : ホイールを回す

・d.MouseDoubleClick : ダブルクリック

・クリックした位置に黒点を打つ

PointF clicked = **new** PointF(0, 0);

SizeF circleSize = **new** SizeF(10, 10);

Drawable d = new Drawable();

d.Paint += (s, pe) => {

pe.Graphics.FillEllipse(Colors.Black, **new** RectangleF(clicked - 0.5F \* circleSize, clicked + 0.5F \* circleSize));

};

d.MouseUp += (s, me) => {

**if** ( me.Buttons.HasFlag(Buttons.Primary) ) {

// 左クリック（左利き用設定をしていない場合）のときの処理

// meはクリックをした座標PointFが格納されている

clicked = me.Location;

d.Invalidate();

}

**else** **if** ( me.Buttons.HasFlag(Buttons.Alternate) ) {

// 右クリックのときの処理

}

Content = d;

};

・クリックした位置に指定した色の複数個の点を打つ

// ColorPickerを用いて色を選択する

namespace MouseDemo3 {

class ColoredCircle{

public PointF Center { get; set; } // 中心座標

public Color Color { get; set; } // 色

public ColoredCircle(PointF center, Color col){

Center = center;

Color = col;

}

public void Draw(Graphics g){ // 実際に描画するところ

SizeF circleSize = new SizeF(10, 10);

g.FillEllipse(Color, new RectangleF(Center - 0.5F \* circleSize, Center + 0.5F \* circleSize));

}

}

public class MainForm : Form{

public MainForm(){

Title = "Mouse Demo 3";

MinimumSize = new Size(200, 200);

List<ColoredCircle> ccs = new List<ColoredCircle>(); // リストを作る

Drawable d = new Drawable();

d.Paint += (s, pe) =>{ // 実際に描画するところ

foreach (var cc in ccs) { cc.Draw(pe.Graphics); }

};

ColorPicker colorPicker = new ColorPicker { Value = Colors.Red };

d.MouseDown += (s, me) => // クリックしたときの挙動{

// 点をリストに追加

ccs.Add(new ColoredCircle(me.Location, colorPicker.Value));

d.Invalidate();

};

Content = new StackLayout { // レイアウト

Padding = 5,

Spacing = 5,

HorizontalContentAlignment = HorizontalAlignment.Stretch,

Items = {

colorPicker,

new StackLayoutItem( d, true )

}

};

Menu = new MenuBar();

}

}

}

**○標準ライブラリ**

・使い方 「型or変数名.メソッド名()」

・出力 Console.WriteLine(値);

値にはオブジェクト型が採用されているのでint, double, bool型は勝手にToString()される。基本値にはstring型を入れる

例. Console.WriteLine(“Hello World!”);

複合書式を使うと便利に記述できる。””の中に{番号}を入れて後から「,」 区切りにする

Console.WriteLine(“{0} \* {1} = {2}”, 10, 3, 10\*3); // 10 \* 3 = 30

・入力 Console.ReadLine();

string s = Console.ReadLine(); // ともできる

・Console.Write(値); だと最後に改行が入らない

・int形のみの入力 Cosole.Read();

・string形への変換 変数名.ToString();

・数学的なこといろいろ Math.メソッド名(値);

例. π → Math.PIE, sinx → Math.sin(x), log10x → Math.log10(x)

・自分のPCのファイルにアクセスするときは現在のディレクトリを指し示してあげる

Environment.CurrentDirectory = Environment.GetFolderPath

(Environment.SpecialFolder.MyDocuments); // 現在ディレクトリを「ドキュメント」に

**○string型に使ういろんな関数**

・配列を1つのstring型変数にして、間に区切りをつける

string s = string.Join(“区切り”, 配列変数)

string[] fruits = {“apple”, “orange”, “banana”};

string one\_fruit = string.Join(“|”, fruits); // apple|orange|banana

・1つのstringを区切りごとに配列に分割する string[] v = stirng型変数.Split(“区切り”);

string[] fruits2 = one\_fruit.Split(“|”); // 3要素の配列になる

・Split()と中身なしだと空白、改行で区切って配列を作ってくれる

・文字列が指定した文字列から始まるときにtrueを返す 元の文字列.StartWith(頭文字)

“おはよう”.StartWith(“おは”) // true

・指定した文字列sが最初から何番目に現れるかを示す。ない場合は-1を返す。

元の文字列.IndexOf(s)

“おはよう”.IndexOf(“はよ”) // 1

・指定した文字の位置以降の文字列を切り出す 元の文字列.Substring(数字)

“ABCDEFGHIJK”.Substring(3) // DEFGHIJK

・位置から「数字2」個分の文字列を切り出す 元の文字列.Substring(数字1, 数字2)

“ABCDEFGHIJK”.Substring(2, 4) // CDEF

・先頭と終端から空白を削除する。ただし一時的で終わったら元に戻る。 変数.Trim()

“ abc ”.Trim() // “abc”

○**オブジェクト**

・クラスを元に実際に作ったもの、同じクラスに何個でも作れて別々に動作する。

・オブジェクトの作成

クラス名 変数名 = new クラス名(引数)

例. Counter c = new Counter(0);

**○修飾子**

・private : それを含むクラスの中からしかアクセスできない

・public : どこからでもアクセスできる

・protected : 自身と派生クラスのみからアクセスできる

・internal : 同一コンパイル単位の中からアクセスできる

・static :

1 2

**○例外処理**

・エラーが起きるときにC#では例外処理を行い、エラーを回避することができる

・try-catch文を用いる。tryには実際のプログラムを書き、catchには例外処理をパターンマッチングで書く。

try{ 本文 }

catch(例外の名前 e){ }

・例外の種類によって名前が異なり、eの中にはエラーメッセージがstring型で入っている

catch(ArgumentNullException e){ Console.WriteLine(e); } // エラーメッセージの表示

例外の種類

|  |  |
| --- | --- |
| Exception | 例外全てを包含している |
| ArgumentNullException | dict(key)のkeyがnullのとき |
| KeyNotFoundException | dict(key)のkeyがまだ登録されていないとき |
| ArgumentException | 自分で設定する例外 |

・throw new ArgumntException(“エラーメッセージ”); // 自分で例外を設定できる

**○C言語と同じところ**

・int, long, float, double型

・宣言文 int a, b = 0, 0;

・式文 a ++;

・return

・for文, while文

・continue, break

**○C言語と異なるところ**

・if文単独のときは必ず{}で囲む。if else文のときは囲まなくてもいい。条件文は1, 0ではtrue, falseを判別できないから注意。

例. if(a > b){ return a; }