SKRYPT

Framework .NET – wszystko razem, czyli całe programowanie w C# będzie pobierać i przenosić wszystko poprzez środowisko .Net, szkielet aplikacji (np. .NET, Unity)

MVVM – model(dane), view(UI), view model (logika), 3warstwowy model aplikacji – UI, logika, dane

VPF – windows presentation foundation (wspiera MVVM)

CLR common language runtime – środowisko uruchomieniowe dla .NET, jest odpowiedzialny za powołanie aplikacji do życia i zarządzanie nią, czyli wszystko znajduje się w .NET a CLR pobiera i przenosi wszystko, wykonuje instrukcje dla IL

FCL framework class library biblioteka wszystkich funkcji z których korzysta CLR, podbiblioteka to BCL

Kompilator zmienia język wysokiego typu na język niskiego typu (ten proces nazywa się kompilacją), którym posługuje się asembler. Asembler zmienia swój język na kod binarny (o i 1) (proces zamiany to też asembler)

Środowisko programistyczne – np. Visual Studio, możesz pisać kody w zwykłym edytorze testu, np. notatnik, ale środowisko programistyczne jest po to żeby ułatwić tą pracę

MSIL albo IL microsoft intermediate language (język pośredni) definiuje instrukcje dla CLR

KOD w C# -> jest kompilowany przez kompilator -> microsoft intermediate language MSIL albo IL

Int (inte ger / ang liczba całkowita) czyli kompilator domyślnie ustawia funkcję na liczby całkowite. Możesz tu określić, że zmienna na jakiej będziesz pracować będzie liczbą całkowitą

ZMIENNE

1. int – liczba całkowita
2. byte – mała zmienna
3. short – liczba całkowita krótka
4. long – liczba całkowita długa
5. float – liczba zmiennoprzecinkowa pojedynczej precyzji
6. double – liczba zmiennoprzecinkowa podwójnej precyzji
7. bool – przechowuje tylko wartość prawda lub fałsz
8. char – odnosi się do pojedynczych znaków lub kodów
9. string – można tu przypisywać całe ciągi znaków
10. var – jedyna zmienna którą trzeba od razu zainicjalizować, ponieważ sama w sobie nie ma typu, a przyjmuje typ jaki zadeklarujesz. Np. jeśli przypiszesz jej ”jhbfbdkfa” to deklaracja będzie string, możesz potem jej przypisywać inne wartości ale nie inny typ. Może być zadeklarowany tylko wewnątrz jakiejś metody, ale na poziomie klasy

ZAKRES ZMIENNYCH

public – zmienna zadeklarowana publicznie, czyli nawet osoby postronne mogą z niej korzystać

internal – zmienna zadeklarowana wewnątrz konkretnego projektu

private – zmienna zadeklarowana wewnątrz konkretnych nawiasów klamrowych, jest to domyślna deklaracja dla zmiennych zadeklarowanych w każdej klasie, wiec nie trzeba tego pisać

Console.WriteLine("TEKST"); - (skrót cw tab tab ) w kodzie widnieje tekst, który jest wpisany pomiędzy ” ”

Console.ReadLine(); - użytkownik wprowadza jakiś ciąg znaków i kompilator go pobiera

string (jeśli chcesz zapisać jakiś ciąg znaków, a będzie Ci on potrzebny jeszcze później to żeby nie pisać go na nowo określasz go na początku jako string)

zmiennaTekstowa (sam określasz jak będzie nazywał się ten ciąg znaków)

int zmiennaLiczbowa; to nazywamy deklaracją zmiennej, czyli nazywasz tą zmienną jak chcesz

zmiennaLiczbowa = 5; to nazywamy inicjalicjacja zmiennej, czyli przypisujemy wartość do zmiennej

Można to zrobić w jednym wierszu: int zmiennaLiczbowa = 5;

bool zmiennaLogiczna = false;

double zmiennaNiecalkowita = 5.76;

zmiennaLiczbowa = zmiennaLiczbowa + 1; // inkrementacja

zmiennaLiczbowa = zmiennaLiczbowa + 5;

zmiennaLiczbowa++; //zmiennaLiczbowa = zmiennaLiczbowa + 1;

string s = "\_\_ala ma kota\_\_\_";

int a= s.Length;

Console.WriteLine(a); - odczytuje ilość znaków w stringu

string b = s.Substring(4,2); - odczytuje 2 znaki po 5 znaku (liczony od 0)

string d = s.Replace("ma", "będzie miała"); // zmienia słowo na inne słowo

string d = s.Replace("\_", ""); - zmienia znak na inny znak

bool e= s.Contains("ma"); -jeśli string ma w sobie dane słowo to wyskakuje true, albo false

string f= s.ToUpper(); - zamienia wszystkie litery na wielkie

string g= "dom";

string h = "Dom";

bool i = g.ToUpper() == h.ToUpper();

int wynik = 10;

string odpowiedz2 = string.Format("policzony wynik: {0} to bardzo mało", wynik); - taki zapis daje „policzony wynik: 10 to bardzo mało” czyli w miejsce {0} wstawia zmienną(w tym przypadku int wynik)

**PROSTY KALKULATOR DODAJĄCY**

Console.WriteLine("Proszę podaj pierwszą liczbę");

int liczba1 = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Proszę podaj drugą liczbę");

int liczba2 = Convert.ToInt32(Console.ReadLine());

int wynik = liczba1 + liczba2;

Console.WriteLine("wynik to " + wynik); // konwersja niejawna

// string wynikString = wynik;

string wynikString = Convert.ToString(wynik); // konwersja jawna

Console.WriteLine("liczba cyfr to: " + wynikString.Length);

To jest zapis do stworzenia prostego kalkulatora

int wzrost = 180

if (!(wzrost > 185)) - ! oznacza że dodatkowo zaprzeczasz wartości która jest w nawiasie // w nawiasie piszesz warunek

{

Console.WriteLine("niski"); // co ma się stać jeśli warunek zostanie spełniony

}

Else if – if piszesz gdy chcesz, żeby był więcej niż jeden warunek

{

Console.WriteLine("wysoki"); // co się stanie jeśli warunek nie zostanie spełniony

}

else if możesz dopisywać kolejne warunki jeśli poprzedni został spełniony ale wciąż go chcesz rozszerzać (lub dodatkowo ograniczać)

else if

else

**OPERATORY**

bool a = 3 == 5; // czy 3 równa się 5

bool b = 5 > 3;

bool c = 5 >= 5; // true

bool d = 5 != 6; // true

bool e = !(3 > 2); // false

bool f = 2 > 1 && 4 > 5; // prawda i fałsz = fałsz

bool g = 5 == 7 || 5 > 3; // fałsz lub prawa = prawda

bool h = 1 > 2 || 3 == 6 || 10 / 2 == 5; // fałsz lub fałsz lub prawda = prawda

bool i = 1 == 1 ^ 2 == 2; // prawda xor prawda = fałsz

if (1 > 2 || 3 == 6 || 10 / 2 == 5)

{

Console.WriteLine("tak");

}

int t = 1;

if (t++==1)

{

Console.WriteLine("tak");

}

else

{

Console.WriteLine("nie");

}

**Solucja prostego kalkulatora**

Console.WriteLine("Podaj równanie");

string rownanie = Console.ReadLine();

int liczba1 = Convert.ToInt32(rownanie.Substring(0,1));

int liczba2 = Convert.ToInt32(rownanie.Substring(2));

string znak = rownanie.Substring(1,1);

if (znak == "+")

Console.WriteLine(liczba1 + liczba2);

if (znak == "-")

Console.WriteLine(liczba1 - liczba2);

if (znak == "\*")

Console.WriteLine(liczba1 \* liczba2);

if (znak == "/")

Console.WriteLine(liczba1 / liczba2);

**SWITCH**

Console.WriteLine("Podaj liczbę");

int liczba = Convert.ToInt32(Console.ReadLine());

switch (liczba)

1:

Console.WriteLine("jeden");

break;

case 2:

Console.WriteLine("dwa");

break;

case 3:

Console.WriteLine("trzy");

break;

case 4:

Console.WriteLine("cztery");

break;

case 5:

Console.WriteLine("piec");

break;

default:

Console.WriteLine("inna");

break;

**PĘTLE**

int length = 10;

int licznik = 0;

int[] liczba = { 1, 3, 5, 6, 8, 3, 2, 5, 7, 888, 883, 5, 22 };

for (int i = 0; i < length; i++)

{

Console.WriteLine("Licznik to: " + i);

}

while (licznik < 5) // w pierwszej kolejności sprawdza warunek, a następnie są wykonywane instrukcje w pętli

{

Console.WriteLine("licznik to: " + licznik);

licznik++;

}

do //w pierwszej kolejności są wykonywane instrukcje w pętli a potem sprawdza warunek, czyli mamy gwarancję że choć raz instrukcje zostaną wykonane

{

Console.WriteLine("Kolejne zadanie: " + licznik);

licznik++;

} while (licznik < 5);

foreach (var item in liczba) //służy do operowania na elementach kolekcji (czyli w pracy np na tablicy, albo listy), nie można zmieniać danych po których iterujemy, dane są dostępne tylko do odczytu

//var + pobieram jeden element (item, ale można nazwać jak chcesz) z kolekcji (w tym przypadku kolekcja to tablica o nazwie liczba

{

Console.WriteLine("Wartość liczby = " + item);

}

**ZADANIE** WYPISZ DOWOLNE ZDANIE, PODAJ DOWOLNY ZNAK ZAWIERAJĄCY SIĘ W TYM ZDANIU I POLICZ ILE RAZY TEN ZNAK ZAWIERA SIĘ W TYM ZDANIU

Console.WriteLine("Podaj zdanie");

string zdanie = Console.ReadLine();

Console.WriteLine(zdanie.Length);

Console.WriteLine("podaj szukany znak");

string znak = Console.ReadLine();

int i = 0;

int ilosc = 0;

while (i < zdanie.Length )

{

if (znak == zdanie.Substring(i, znak.Length)) //jeżeli znak(a) zawiera się w każdym kolejnym znaku zdania to zwiększ zmienna ilosc o jeden

{

ilosc++;

}

i++; //zwiększaj się dopóki zdanie się nie skończy

}

Console.WriteLine(ilosc); // wypisz ostateczną warość ilość zwiększoną tyle razy ile razy spełnił się warunek

**ZADANIE** UZYTKOWNIK SAM PODAJE KILKA ELEMENTÓW TABLICY, POTEM SYSTEM WYŚWIETLA KTÓRY ELEMENT JEST NAJMNIEJSZY

Console.WriteLine("proszę o podanie ile elementów chcesz przechowywać");

int k = Convert.ToInt32(Console.ReadLine());

int[] tab = new int[k];

int i = 0;

while (i < k)

{

tab[i] = Convert.ToInt32(Console.ReadLine());

i++;

}

int najmniejszy = tab[0];

i = 1;

while (i < k)

{

if (tab[i]<najmniejszy)

najmniejszy = tab[i];

i++;

}

Console.WriteLine("najmniejszy: " + najmniejszy);

**LUB**

Console.WriteLine("Podaj kolejne liczby");

int[] tab1 = new int[5];

tab1[0] = Convert.ToInt32(Console.ReadLine());

tab1[1] = Convert.ToInt32(Console.ReadLine());

tab1[2] = Convert.ToInt32(Console.ReadLine());

tab1[3] = Convert.ToInt32(Console.ReadLine());

tab1[4] = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Ktora wartosc jest najmniejsza?");

int najmniejszy = tab1[0];

int i = 1;

while (i < 5)

{

if (tab1[i] < najmniejszy) //jeżeli iTY (czyli w tym przypadku 1 (czyli 2 według poprawnego wyliczania) element tablicy jest mniejszy niż 0 (czyli 1)

najmniejszy = tab1[i]; // jeżeli wyrażenie logiczne to prawda to wyświetl iTY element

i++; // zwiększ iTY element o 1, po to żeby pętla mogła sprawdzić If dla elementu kolejnego

}

**LISTY**

List<string> lista = new List<string>();// zadeklaruj nową listę

int dl = lista.Count; // policz liczbę elementów listy

lista.Add("ala"); // dodaj element do listy

lista[0] = "ala2"; // zadeklaruj element 1 z listy

string s = lista[0]; //zadeklaruj string który będzie 1 elementem listy

lista.Add("ma");

lista.Add("kota");

lista.Remove("kota"); //usuwa konkretny elemet listy

lista.RemoveAt(1); //usuwa konkretny elemet który jest na pozycji ... np 2

string[] tab= lista.ToArray(); // zadeklaruj nową tablicę, której elementy będą elementami listy

List<string> lista2 = tab.ToList(); // zadeklaruj nową listę, której elementy będą elementami tablicy

List<long> nowaLista = new List<long>();

nowaLista.Add(200);

nowaLista.Add(40);

nowaLista.Add(43534);

nowaLista.Add(23542);

nowaLista.Add(756);

//modyfikacja wartości elementu listy na określonej pozycji

nowaLista[0] = 1;

//dodanie elementu listy na określonej pozycji

nowaLista.Insert(0, 872);

//usunięcie elementów listy, które spełniają określone warunki

nowaLista.RemoveAll(x => x > 1000);

//ususięcie konkretnego elementu listy

nowaLista.RemoveAt(0);

foreach (var item in nowaLista)

{

Console.WriteLine(item);

}

Console.ReadKey();

**ZADANIE** PODAJ DOWOLNĄ LICZBĘ IMION ODDZIELONYCH ENTEREM, A POTEM WYŚWIETL KAŻDY ELEMENT POWSTAŁEJ LISTY WRAZ Z CYFRĄ SYMBOLIZUJĄCĄ LICZBĘ LITER W KAŻDYM IMIENIU

Console.WriteLine("Podaj kolejno pracowników");

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

string wczytaneImie = Console.ReadLine(); //deklarujesz string który będzie wczytywał tekst

imiona.Add(wczytaneImie); //poprzedni string jest dodawany do listy

while (wczytaneImie != "") //pętla trwa dopóki kolejny wers jest różny od pustego wersa, ale pusty wiersz też zalicza się do listy

{

wczytaneImie = Console.ReadLine(); //w pętli wpisujesz imiona

imiona.Add(wczytaneImie); // każde z imion jest dodawane do pętli

}

imiona.RemoveAt(imiona.Count - 1); //usuwa pusty wiersz z listy

int i = 0; // standardowa deklaracja iTego elementu pętli while

while (i<imiona.Count) //pętla trwa tak długo jak liczba elementów listy

{

Console.WriteLine(**string.Format("{0} ({1})",**imiona[i],imiona[i].Length)); //wyświetl iTY element listy w określony sposób

i++;

}

INNE ROZWIĄZANIA TEGO ZADANIA

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

string imie;

do

{

imie = Console.ReadLine();

listaImion.Add(imie);

} while (imie != "");

// inne rozwiązanie

List<string> listaImion2 = new List<string>();

while (true)

{

string imie2 = Console.ReadLine();

if (imie2 == "")

break; // wychodzi z aktualnej iteracji i całej pętli

listaImion2.Add(imie2);

}

JAK WYŚWIETLAĆ ELEMENTY Z LISTY

List<string> imiona = new List<string>() { "Adam", "Lukasz", "Ala", "Marian" };

int i = 0;

while (i<imiona.Count)

{

Console.WriteLine(imiona[i]);

i++;

}

for (int j = 0; j < imiona.Count; j++)

Console.WriteLine(imiona[i]);

**TABLICE**

TABLICA jeśli w [ ] jest napisane 3 czyli [3] to musi w tablicy być 3 elementów, ani jednego mniej czy więcej. Napisanie [ ] bez liczby w środku powoduje że tablica nie musi mieć konkretnej liczby elementów, ale domyślnie ma tyle elementów ile wpiszę.

**DOSTĘP DO PLIKÓW**

File.WriteAllText(@"c:\pliki\plik.txt", "ala ma kota");

string s= File.ReadAllText(@"c:\pliki\plik2.txt");

string[] tab= File.ReadAllLines(@"c:\pliki\plik2.txt");

Console.WriteLine(s);

string wynik = string.Format("{0} {1} ({2})", komorki[2], komorki[3], komorki[4]); //wyświetla napi w określonej formie, np {1} spacja(też jest ważna), w miejsce 1 wsawiamy dowolny element

string kraj = Console.ReadLine().ToUpper();// cokolwiek wpisze użytkownik będzie zamienione na wielkie litery

**konkatenacja** znak „+” może być wykorzystywany również do łączenia wartości różnych typów w jeden ciąg wynikowy, np.:

double e = 2.33;

string f = "zł";

Console.WriteLine("cena towaru to " + e + " " + f);

WEBClient korzystanie z kodu stron internetowych

string url = "https://www.wp.pl/";

string dane = new WebClient().DownloadString(url);

Console.WriteLine(dane); //powoduje wyświetlenie kodu strony internetowej w consoli

File.WriteAllText(@"C:\Users\Sebastian\Desktop\01 Programowanie\Pliki\strona.html", dane); // zapisuje stronę internetową jako plik na kompline

uterze

Console.ReadKey(); - wyświetla wszystko co było do tej pory zapisane w kodzie

**KLASY Programowanie obiektowe:** dzięki temu można dzielić kod na konkretne fragmenty

Na poziomie klasy nie pisze się odwołań w stylu console.writeline bo może to nie zadziałać we wszystkich kompilatorach.

Klasy tworzy się głównie dla innych programistów a nie dla użytkowników końcowych

Private w konstruktorach pisze się po to, żeby nie można było go ruszać, ja jako programista ustalam to sobie i inni nie mogą jej zmieniać, public wręcz przeciwnie, można z niej korzystać i ją modyfikować

ATRYBUTY KLASY (przyjęło się, że ta kolejność jest poprawna):

1. Pola -- cechy danego obiektu, np. waga, kolor, dostajemy się do tego poprzez zmienną, np:

Czlowiek c1 = new Czlowiek();

W klasie: public string Imie;

c1. (ciemnoniebieski prostopadłościan w inteisens w metodzie main)

1. Właściwości (property)-- to samo co pole, tylko bardziej rozbudowane

(czarny klucz w inteisens w metodzie main)

3) Konstruktory -- określamy tutaj w jaki sposób tworzymy obiekt, np. dzięki temu mogę wymusić, że jeśli ktoś stworzy obiekt NEW CZŁOWIEK, to musi zadeklarować jego imię i nazwisko

4) Metody -- możliwości danego obiektu , czyli co obiekt potrafi zrobić, np. człowiek może iść, kalkulator może coś policzyć.

(niebieski przeźroczysty sześcian w inteisens w metodzie main)

Jak się pisze metodę: np public int PodajTemperaturę(string miasto)

Public / private publiczna czy prywatna

Int / string / void itp. Każda metoda musi coś zwracać, czyli metoda się wykonuje i zwraca coś na zwewnątrz (piszemy wtedy public, lub private), lub metoda się wykonuje ale nic nie zwraca na zewnątrz (wtedy piszemy void)

PodajTemperaturę czyli nazwa

protected pozwala na korzystanie z tej funkcji w tej klasie oraz w każdej klasie która po niej dziedziczy

() w nawiasie podajemy argumenty metody, po przecinku każdy następny argument

INSTANCJA: gdy zadeklarujemy nową klasę, np. samochód, to nowa instancja to będzie np. samochód polo, samochód ww, samochód mercedes

symbol Entera: ⏎ — najpierw linia w dół (\n), potem powrót karetki (\r), tak jak w maszynach do pisania.

double bmi = waga / Math.Pow(wzrost/100.0, 2); co powoduje?

Data banding = wiązanie danych, kontrolki są powiązane z konkretnymi danymi

**BAZY DANYCH**

Viev -> Serwer Explorer-> data conection->add conection->Microsoft SQL Server->serwer name (tu musi być IP serwera) np. mssql4.webio.pl,2401 -> authentication (windows authentication, czyli loguje się jako użytkownik zalogowany na Windowsie, SQL Server authentication loguje się jako użytkownik zalogowany na koncie stworzonym w SQL, najczęściej się to stosuje gdy server nie jest w naszej przestrzeni lokalnej) -> user name and password->

Najważniejsze po zalogowaniu to zakładka tables

Views to widok na podzbiór danych (zamiast na całą tabelkę)

Stored Procedures: operacje na bazach danych

Funcions: możesz definiować swoje własne funkcje

Synonims: nazwy alternatywne do nazw oryginalnych elementów, np. nazwać tabelę po swojemu, gdy łączymy się z jakąś tabelą zewnętrzną, która ma swoją nazwę.

Types / Assemblers: własne typy i odwołania

Otwieranie bazy danych: show open data

select \* from zawodnicy podstawowe polecenie (ctrl+shift+e

RÓŻNICA POMIĘDZY METODĄ A WŁAŚCIWOŚCIĄ

WŁAŚCIWOŚĆ na końcu nie podaje się nawiasu

METODA na końcu musi mieć nawias

CompareTo jeśli 1 element jest „większy” to zwraca 1, jeśli jest równy to 0, jeśli mniejszy to -1, w przypadku stringów „większy” oznacza późniejszą literę.

Rename ctrl +R zmienia nazwę we wszystkich miejscach

Uzytkownik?.Imie jeżeli „Uzytkownik” jest null to wyświetli się „null” a jeśli będzie miał wartość to wyświetli się wartość.

Ctrl H – zaznacz część tekstu a potem napisz co chcesz podmienić

Funkcja Mock, jest taką wydmuszką, niby jest tą samą funkcją, którą wywołuje, ale jednocześnie może wyciągnąć jeden element funkcji, wstawić tam swoje elementy i je przetestować

Interfejsy: doczytaj co to

Metoda anonimowa: nie ma żadnej definicji nazwy

() => (operator lambda) dla metody która nie przyjmuje żadnego parametru, patrz dalej definicję tej metody

Co robią?

ItemsSource=skąd pobierane są dane

SelectedItem=jeśli coś kliknę na grid (np. szczegóły na liście) to w tym miejscu zasciągną się dane z tego konkretnego wiersza (np. selectedClient)