Programowanie II

Lista 5

Programowanie obiektowe

Co to jest w ogóle klasa ?

```
/*
Czym jest klasa - to prostu definicja nowego typu
taki typ może łączyć składniki różych typów (w tym innych klas) oraz funkcji
(od tego momentu nazywanych metodami) działających na tych składnikach,
*/

#include "pch.h"
#include <iostream>
 using namespace std;

class TCzlowiek //definicja nowej  klasy o nazwie TCzlowiek
{
public:
    int wzrost;
    int waga;
    /*
    klasa posiada dwa składniki (atrybuty), oba składniki są publiczne, tj. dostępne
    dla wszystkich elementów programu
    */
};

void WyswietlCzlowieka(TCzlowiek Wyswietlany)
//tak więcej nie będizemy robić - funkcje staną sie metodami klas
{
   cout << "********************" << endl;
   cout << "Wzrost: " << Wyswietlany.wzrost << endl;
   cout << "Waga: " << Wyswietlany.waga << endl;
   cout << "********************" << endl;


}
```

```
int main()
{
   TCzlowiek Europejczyk;
   /*
   Tworzymy nowy obiekt klasy TCzlowiek - można powiedzieć, że deklaruejmy zmienną typu TCzlowiek
   */
   Europejczyk.waga = 75;
   Europejczyk.wzrost = 175;
   //nadajemy wartości atrubtom obiektu Europejczyk
   WyswietlCzlowieka(Europejczyk);

   TCzlowiek Azjata; //nowy obiekt klasy TCzlowiek
   TCzlowiek *WskNaCzlowieka; //wskaźnik na obiekty klasy TCzlowiek
   WskNaCzlowieka = &Azjata; //do wskaźnika przypisujemy adres obiektu Azjata
   WskNaCzlowieka->waga = 68; // odwołując się do składników klasy za pośrednictwem wskaxnika używamy "->"
   WskNaCzlowieka->wzrost = 165;
   WyswietlCzlowieka(*WskNaCzlowieka); //funkcja oczekuje na obiekt nie na wskaźnik
   system("pause");
}
```

Klasa w pliku nagłówkowym – TCzlowiek.h

```
1   #pragma once
2   class TCzlowiek //definicja nowej  klasy o nazwie TCzlowiek
3   {
4   public:
5       int wzrost;
6       int waga;
7       /*
8       klasa pozisada dwa składniki (atrybuty), oba składniki są publiczne, tj. dostępne
9       dla wszystkich elementów programu
10      */
11  };
```

```
3   #include "pch.h"
4   #include "TCzlowiek.h"
5   //plik nagłówkowy zawierający definicję klasy TCzlowiek
6   #include <iostream>
7   using namespace std;
8
9
10
11  void WyswietlCzlowieka(TCzlowiek Wyswietlany)
12  {
13      cout << "********************" << endl;
14      cout << "Wzrost: " << Wyswietlany.wzrost << endl;
15      cout << "Waga: " << Wyswietlany.waga << endl;
16      cout << "********************" << endl;
17
18  }
19  int main()
20  {
21      TCzlowiek Europejczyk;
22      /*
23      Tworzymy nowy obiekt klasy TCzlowiek - można powiedzieć, że deklaruejmy zmienną typu TCzlowiek
24      */
25      Europejczyk.waga = 75;
26      Europejczyk.wzrost = 175;
27      //nadajemy wartości atrubtom obiektu Europejczyk
28      WyswietlCzlowieka(Europejczyk);
29
30      TCzlowiek Azjata; //nowy obiekt klasy TCzlowiek
31      TCzlowiek *WskNaCzlowieka; //wskaźnik na obiekty klasy TCzlowiek
32      WskNaCzlowieka = &Azjata; //do wskaźnika przypisujemy adres obiektu Azjata
33      WskNaCzlowieka->waga = 68; // odwołując się do składników klasy za pośrednictwem wskaznika używamy "->"
34      WskNaCzlowieka->wzrost = 165;
35      WyswietlCzlowieka(*WskNaCzlowieka); //funkcja oczekuje na obiekt nie na wskaźnik
36      system("pause");
37  }
```

Metody klasy – TCzlowiek.h

```
1   #pragma once
2   #include "pch.h"
3   #include <iostream>
4   using namespace std;
5
6   class TCzlowiek //definicja nowej  klasy o nazwie TCzlowiek
7   {
8   public:
9       int wzrost;
10      int waga;
11      /*
12      klasa pozisada dwa składniki (atrybuty), oba składniki są publiczne, tj. dostępne
13      dla wszystkich elementów programu
14      */
15      void WyswietlSie()
16      /*
17          Funkcja staje się metodą - nie wyświetlamy jakiegoś człowieka tylko samego siebie
18      */
19      {
20          cout << "********************" << endl;
21          cout << "Wzrost: " << wzrost << endl;
22          //do atrybutów możemy odwoływać się wprost
23          cout << "Waga: " << this->waga << endl;
24          //albo za pomocą tzw. wskaźnika na samego siebie - popularny w innych językach
25          cout << "********************" << endl;
26
27      }
28  };
```

```
3   #include "pch.h"
4   #include "TCzlowiek.h"
5   #include <iostream>
6   using namespace std;
7
8
9
10  int main()
11  {
12      TCzlowiek Europejczyk;
13      Europejczyk.waga = 75;
14      Europejczyk.wzrost = 175;
15      Europejczyk.WyswietlSie(); // wywołujemy metodę wyświetl się
16      TCzlowiek Azjata;
17      TCzlowiek *WskNaCzlowieka;
18      WskNaCzlowieka = &Azjata;
19      WskNaCzlowieka->waga = 68;
20      WskNaCzlowieka->wzrost = 165;
21      WskNaCzlowieka->WyswietlSie(); // wywołujemy metodę wyświetl się
22      system("pause");
23  }
```

Zasłanianie nazw

```
/*
Zasłanainie nazw - w związku z tym, że nazwy składników klasy mają zakres klasy
to w obrębie klasy zasłaniają elementy o takich samych nazwach leżących poza klasą
*/
#include "pch.h"
#include <iostream>
using namespace std;

int ZmiennaA = 1;

class JakasKlasa
{
public:
    int ZmiennaA = 2; // ta zmienne przesłania zmienną globalną
    void JakasMetoda()
    {
        int ZmiennaA = 3; //zmienna lokalna metody
        cout << "zmienna lokalna metody: " << ZmiennaA << endl;
        cout << "zmienna lokalna metody: " << this->ZmiennaA << endl;
        // dzieki this możemy odwołać się do przesłoniętego przez zmienną lokalną składnika klasy
    }
};

int main()
{
    JakasKlasa ObiektJakiejsKlasy;

    cout <<"zmienna globalna: "<<ZmiennaA << endl; //zmienna globalna
    cout << "zmienna jako skladnik klasy: " <<ObiektJakiejsKlasy.ZmiennaA << endl;
    /*
    Zmienne jako atrybut klasy
    */
    ObiektJakiejsKlasy.JakasMetoda();
    system("pause");
}
```

Konstruktor

```
1   /*
2   Konstruktor - specjalna metoda, która jest wywoływana automatycznie
3   w momencie tworzenia nowego obiektu danej klasy. Jej specyfika
4   polega na tym, ze jako jedyna nie zwraca żadnego typu, zatem
5   nie używamy tutaj nawet void, a dodatkowo nazywa się literalnie
6   dokładnie tak samo jak brzmi nazwa klasy. Jak łatwo się domyślić
7   - klasa może mieć więcej konstruktorów - muszą jedna różnić się
8   liczbą parametrów.
9   */
10
11  #include "pch.h"
12  #include <iostream>
13  #include <string>
14  using namespace std;
15
16  class TStudent
17  {
18  public:
19      string Uczelnia = "Nieokreslona";
20
21      void PrzedstawSie()
22      {
23          cout << "Uczelnia: " << Uczelnia << endl;
24      }
25  };
26  class TStudent2
27  {
28  public:
29      string Uczelnia = "Nieokreslona";
30
31      void PrzedstawSie()
32      {
33          cout << "Uczelnia: " << Uczelnia << endl;
34      }
35
36
37      TStudent2(string NUczelnia)
38      {
39          Uczelnia = NUczelnia;
40      };
41  };
42
```

```
43  class TStudentPWSZwNysie
44  {
45  public:
46      string Imie;
47      string Nazwisko;
48      string Uczelnia;
49      string Kierunek;
50      TStudentPWSZwNysie()// własny konstruktor bezparametrowy
51      {
52          Uczelnia = "PWSZ w Nysie";
53          Kierunek = "nie wskazano";
54          Imie = "nie okreslono";
55          Nazwisko = "nie okreslono";
56      }
57      TStudentPWSZwNysie(string Nkierunek) // konstruktor z dwoma paramterami
58      {
59          Uczelnia = "PWSZ w Nysie";
60          Kierunek = Nkierunek;
61          Imie = "nie okreslono";
62          Nazwisko = "nie okreslono";
63      }
64
65      TStudentPWSZwNysie(string Nkierunek, string NNazwisko, string NImie);
66      // definicja konstruktora z 3 parametrami
67      void PrzedstawSie();
68      //definicja metody
69
70  };
71
72  TStudentPWSZwNysie::TStudentPWSZwNysie(string Nkierunek, string NNazwisko, string NImie)
73  //ciało konstruktora 3 param. umieszczone poza klasą
74  {
75      Uczelnia = "PWSZ w Nysie";
76      Kierunek = Nkierunek;
77      Imie = NImie;
78      Nazwisko = NNazwisko;
79  }
80
81  void TStudentPWSZwNysie::PrzedstawSie()
82  {
83      cout << "*********************************"<<endl;
84      cout << "imie: " << Imie << endl;
85      cout << "nazwisko: " << Nazwisko << endl;
86      cout << "uczelnia: " << Uczelnia << endl;
87      cout << "kierunek: " << Kierunek << endl;
88      cout << "*********************************" << endl;
89  }
```

```
90
91  int main()
92  {
93      TStudent Kowalski;
94      //wykorzystaliśmy domyślny konstruktor bezparamatrowy
95      Kowalski.PrzedstawSie();
96      //TStudent2 Kowalski2;
97      /*
98      POwyższy wiersz powoduje błąd - klasa już ma inny konstruktor - z parametrem,
99      zatem nie możemy używać domyślnego - oczywiśćie n
100     */
101     TStudent2 Kowalski2("PWSZ w Nysie");
102     Kowalski2.PrzedstawSie();
103     /*
104     Tutaj korzystamy z naszego konstruktora parametrowego - student od razu ma
105     konkretnie ustaloną nazwę uczelni
106     */
107     TStudentPWSZwNysie Kowalski3;
108     Kowalski3.PrzedstawSie();
109     TStudentPWSZwNysie Kowalski4("Informatyka");
110     Kowalski4.PrzedstawSie();
111     //konstrukor z 1 parametrem
112     TStudentPWSZwNysie Kowalski5("Informatyka","Kowalski","Tomasz");
113     Kowalski5.PrzedstawSie();
114     //konstruktor z 3 parametrami
115     TStudentPWSZwNysie Kowalska{ "Pielegniarstwo","Kowalska","Anna" };
116     //można też przy użyciu nawiasów blokowych
117     TStudentPWSZwNysie *WskNaStud;
118     WskNaStud = new TStudentPWSZwNysie("Finanse", "Nowak", "Jan");
119     WskNaStud->PrzedstawSie();
120     /*
121     mozemy operować na studencie nie tworząc obiektu wprost - można posługiwać się
122     wskaźnikiem. Oczywiście w takim przypadku do wskaźnika musimy przypisać wynik
123     działania metody new, która do zaalokowanej pamięci wpisze efekt działania
124     jawnie wywołanego konstruktora.
125     */
126     system("pause");
127 }
```

Konstruktor kopiujący

```
/*
Konstruktor kopiujący pozwala określić sposób utworzneie nowego
obiektu danej klasy na podstwie innego, wcześniej istniejącego
*/
#include "pch.h"
#include <iostream>
#include <string>
using namespace std;

class TJakasKLasa
{
public:
    int skladnik1;
    int skladnik2;
    TJakasKLasa()
    {};
    TJakasKLasa(int skl1, int skl2)
    //zwykły konstruktor parametrowy
    {
        skladnik1 = skl1;
        skladnik2 = skl2;
    };
    TJakasKLasa(TJakasKLasa &zrodlo)
    {
    //przekazujemuy do kontruktora tylko referencję do oryginału, żeby nie marnowac pamięci
        skladnik1 = zrodlo.skladnik1;
        skladnik2 = zrodlo.skladnik2;
    };
    string toString()
    {
        string wynik = "skladnik 1: " + to_string(skladnik1) + ", skladnik 2: " + to_string(skladnik2) + "\n";
        return wynik;
    }
};
int main()
{
    TJakasKLasa obiekt(2,3);
    cout << obiekt.toString();
    TJakasKLasa obiekt2(obiekt);
    //tworzymy nowy obiekt na podstwie starego
    cout << obiekt2.toString();
    system("pause");
}
```

Destruktor

```
/*
Destruktor to poodbnie jak konstruktor specjalna metoda, która jest wywoływana
tuż przed usunięciem obiektu z pamięci. Jest tworzony automatycznie
(i nie jawnie) dla każdej nowej klasy, ale oczywiście, możemy go napisać
w wersji własnej.
*/
#include "pch.h"
#include <iostream>
#include "TKlasaDetruktor.h"
using namespace std;

int main()
{
    TKlasaDetruktor Obiekt;
    Obiekt.~TKlasaDetruktor();
    system("pause");

}
```

TKlasaDestruktor.h

```
/*
Jeżeli nową klasę dodajemy z zpoiomu VisualStudio do automatycznie
generowane są jawne konstruktory i dekstory bezparametrowe
*/
#pragma once
#include <iostream>
using namespace std;
class TKlasaDetruktor
{
public:
    TKlasaDetruktor();
    ~TKlasaDetruktor();
    //destruktor ma identyczną nazwę jak konstruktor + ~ na początku
};



TKlasaDetruktor::TKlasaDetruktor()
{
    cout << "Wlasnie powstal kolejny obiekt klasy TKLasaDestruktor" << endl;
}


TKlasaDetruktor::~TKlasaDetruktor()
{
    cout << "Obiekt klasy TKlasaDestruktor zaraz zostanie usunięty" << endl;
}
```

Zadanie 1 (30 pkt)

W oparciu o powyższe informacje proszę przygotować zestaw klas, które szczegółowo opiszą wszystkie typy pól występujących w grze planszowej „Monopoly”(Eurobusiness). Oczywiście w zależności od kategorii poszczególnych pól, muszą one przechowywać informacje o kosztach zakupu samej nieruchomości, zakupu domów, hoteli, wysokości opłat, hipotekach, grzywnach, bonusach itp. itd. Zaproponuj następnie klasy realizujące zadania bankiera, graczy itp. Należy wyposażyć klasy w odpowiednie konstruktory oraz szablony najważniejszych metod obsługujących kupowanie, sprzedawanie nieruchomości, naliczanie i pobieranie opłat za postój, stawianie domów i hoteli. Każda metoda powinna mieć precyzyjnie określony nagłówek wraz z potrzebnymi jej do działania parametrami. W ciele metody powinien się pojawić w formie komentarza możliwie precyzyjny opis jej działania, z szczególnym uwzględnieniem tego co się ma dziać z parametrami.