Programowanie II

Lista 7

Modyfikatory dostępu

plik TKLientBanku.h

```
6   class TKlientBanku
7   {
8   public:
9       string Imie;
10      string Nazwisko;
11      //deklaracje metotd
12
13      void SetNrKonta(string NowyNumer);
14      string GetNumerKonta();
15      string SetPin(int NowyPIN);
16      int GetPIN();
17      string SetStanKonta(double NowyStan);
18      double GetStanKonta();
19      string toString();
20      /*
21      Tak zwyczajowo nazywamy metodę, której zadaniem jest przygotowanie
22      opisu obiektu klasy w postaci tekstu.
23      */
24  protected:
25      string NrKonta;
26  private:
27      int PIN;
28      double StanKonta;
29  };
30
31  //definicje metod
32  void TKlientBanku::SetNrKonta(string NowyNumer)
33  {
34      NrKonta = NowyNumer;
35  }
36
```

```
37  string TKlientBanku::GetNumerKonta()
38  {
39      /*
40      tutaj można by np. wprowadzić jakieś zabezpiecznie, które będzie co śweryfikować
41      przed udostępnieniem numeru konta do publicznej wiadomości
42      */
43      return  NrKonta;
44  }
45
46  string TKlientBanku::SetPin(int NowyPIN)
47  {
48      if ((NowyPIN >= 0) && (NowyPIN <= 9999))
49      {
50          PIN = NowyPIN;
51          return "PIN zostal ustawiony";
52      }
53      else
54          return "PIN poza zakresem !";
55  }
56
57  int TKlientBanku::GetPIN()
58  {
59      return PIN;
60  }
61
```

```
62  string TKlientBanku::SetStanKonta(double NowyStan)
63  {
64      string DoZwrotu;
65      if (NowyStan > 0)
66      {
67          StanKonta = NowyStan;
68          DoZwrotu = to_string(StanKonta);
69      }
70      else
71          DoZwrotu = "Stan konta nie moze byc < 0";
72      return DoZwrotu;
73  }
74
75  double TKlientBanku::GetStanKonta()
76  {
77      return StanKonta;
78
79  }
80  string TKlientBanku::toString()
81  {
82      string DoZwrotu = "";
83      DoZwrotu += "Imie: " + Imie + "\n";
84      DoZwrotu += "Nazwisko: " + Nazwisko + "\n";
85      DoZwrotu += "Numer konta: " + GetNumerKonta() + "\n";
86      DoZwrotu += "PIN: " + to_string(GetPIN()) + "\n";
87      DoZwrotu += "Stan konta: " + to_string(GetStanKonta()) + "\n";
88
89
90      return DoZwrotu;
91  }
92
```

plik z funkcją main

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

int main()
{
    TKlientBanku JKowalski;
    JKowalski.Imie = "Jan";
    JKowalski.Nazwisko = "Kowalski";
    //JKowalski.NrKonta = "123456789";
    //tak nie można - nie mamy bezpośredniego dostępu do skladników określonych jako protected.
    JKowalski.SetNrKonta("123456789");
    cout << JKowalski.SetPin(12345) << endl;
    /*
    tutaj wyswietlony zostanie komunikat, że PIN jest poza zakresem
    tj. zadziała zaplanowane zabezpieczenie
    */
    cout << JKowalski.SetPin(4444) << endl;
    cout << JKowalski.SetStanKonta(-100) << endl;
    //tutaj też zadziała zabezpieczenie
    cout << JKowalski.SetStanKonta(100) << endl;
    cout << JKowalski.toString();
    system("pause");
}
```

Przyjaźń

```cpp
/*
Jak wiadomo do prywatnych składowych klasy dostęp mają tylko składniki danej klasy.
Czasami może się jednak okazać, że jakaś funkcja spoza zakresu klasy powinna mieć również dostęp do prywatnych składników klasy.
Aby osiągnąć powyższy efekt wystarczy w definicji klasy umieścić deklarację tej funkcji poprzedzając ją słowem friend.
Należy pamiętać, że to nie funkcja ma twierdzić, że jest zaprzyjaźniona z klasą.
To klasa powinna zadeklarować, że przyjaźni się z daną funkcją i chce udostępnić jej swoje prywatne składniki.
Funkcja może być zaprzyjaźniona z więcej niż jedną klasą
Deklaracja przyjaźni umożliwia dostęp do składników prywatnych nawet takim funkcjom, które ze względów technicznych
nigdy nie mogłyby stać się składowymi klasy (np. dlatego, że były by napisane w innym języku programowania)
*/

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

class konto_bankowe;
class urzad_skarbowy
{
public:
    void kontrola_skarbowa(konto_bankowe konto);
};

```

```
24  class konto_bankowe
25  {
26  public:
27      string numer;
28      string imie;
29      string nazwisko;
30      void ustal_stan_konta(float nowy_stan);
31      float zwroc_stan_konta();
32      friend  void SamodzielnaKontrolaSkarbowa(konto_bankowe konto);
33      friend  void urzad_skarbowy::kontrola_skarbowa(konto_bankowe konto);
34  private:
35      float stan_konta;
36  };
37
38  void konto_bankowe::ustal_stan_konta(float nowy_stan)
39  {
40      stan_konta = nowy_stan;
41  };
42
43  float konto_bankowe::zwroc_stan_konta()
44  {
45      return stan_konta / 3;
46  };
47
48
49
50  void urzad_skarbowy::kontrola_skarbowa(konto_bankowe konto)
51  {
52      cout << "Stan konta o numerze " << konto.numer << ":" << "\n";
53      cout << konto.stan_konta << "\n";
54  };
55
56
57  /*
58  to jest funkcja o zekresie pliku - nie ma nic wspolnego z klasą, ale jest zaprzyjazniona i dlatego
59  ma dostęp do składników prywatnych tej klasy
60  */
61  void SamodzielnaKontrolaSkarbowa(konto_bankowe konto)
62  {
63
64      cout << "Stan konta o numerze " << konto.numer << " :" << "\n";
65      cout << konto.stan_konta << "\n";
66  };
67
68  int main()
69  {
70
71      konto_bankowe kontoKowalskiego;
72      kontoKowalskiego.numer = "234324";
73      kontoKowalskiego.ustal_stan_konta(100);
74      cout << "Stan konta o numerze " << kontoKowalskiego.numer << "\n" << kontoKowalskiego.zwroc_stan_konta() << "\n";
75      urzad_skarbowy urzadNysa;
76      urzadNysa.kontrola_skarbowa(kontoKowalskiego);
77      SamodzielnaKontrolaSkarbowa(kontoKowalskiego);
78      system("pause");
79  }
```

Składniki statyczne

```
23  class TStudent
24  {
25  public:
26      int index;
27      char imie[40];
28      static int ile_studentow; //skladnik statyczny
29  };
30
31  int TStudent::ile_studentow=0;
32  //aby skladniki były dostępne globalnie muszą byc też tak zadeklarowane
33  int main()
34  {
35      TStudent *Student1, *Student2, *Student3;
36      Student1 = new TStudent;
37      Student2 = new TStudent;
38      cout << (Student1->ile_studentow) << "\n";
39      cout << (Student2->ile_studentow) << "\n";
40      //obaj zeznają, ze studentów jest 0
41      Student1->ile_studentow = 2;
42      //Student1 zwiększa wartość składnika statycznego
43      cout << (Student1->ile_studentow) << "\n";
44      cout << (Student2->ile_studentow) << "\n";
45      //obaj zeznają, ze studentów jest 2
46      TStudent::ile_studentow = 3;
47      // korzystamy tylko z określenia zakresu klasy, i zwiększamy ilość studentów
48      cout << (Student1->ile_studentow) << "\n";
49      //Student1 oczywiście stwierdza, że jest ich 3
50      system("pause");
51  }
```

Statyczne funkcje składowe

```
11  class TStudent
12  {
13  public:
14      int index;
15      char imie[40];
16      static int ile_studentow; //skladnik statyczny
17      static void UstawIloscStudentow(int ile)
18      {
19          ile_studentow=ile;
20      }
21  };
22
23  int TStudent::ile_studentow = 0;
24  //aby skladniki były dostępne globalnie muszą byc też tak zadeklarowane
25  int main()
26  {
27      TStudent *Student1;
28      Student1 = new TStudent;
29      Student1->UstawIloscStudentow(1);
30      //wykorzystujemy metodę statyczną
31      cout << (Student1->ile_studentow) << "\n";
32      TStudent::UstawIloscStudentow(2);
33      TStudent *Student2;
34      Student2 = new TStudent;
35      /*
36      Najpierw wykorzystaliśmy zakres klasy do uruchomienia funkcji statycznej,
37      a potem utworzyliśmy nowy obiekt tej klasy - oczywiście on również "wie" o wszystko o
38      aktualnej wartości wszystkich składników statycznych
39      */
40      cout << (Student1->ile_studentow) << "\n";
41
42      system("pause");
43  }
```

Zadanie 1 (10 pkt)

Zmodyfikuj strukturę klas gry monopolly (z listy nr 4) tak aby uwzględnić konieczność zabezpieczenia niektórych pól przed nieautoryzowanym dostępem (zastanów się, które pola tego wymagają, ze względu na przebieg rozgrywki). W konsekwencji zaproponuj odpowiednie deklaracji przyjaźni, aby realizacja zadań np. bankiera była możliwa.

Zadanie 2 (10 pkt)

Zaimplementuj małą klasę, która będzie wyposażona w statyczne metody pozwalające wyznaczać następujące miary dla zbioru liczb całkowitych (reprezentowanego w postaci wektora liczb):

- minimum
- maksimum
- średnia
- mediana
- dominanta