Programowanie obiektowe – język C++

Dr inż. Sławomir Samolej
D108A, tel: 865 1486,
email: ssamolej@prz-rzeszow.pl
WWW: ssamolej.prz-rzeszow.pl

Podziękowanie:

Chcę podziękować dr inż.. Grzegorzowi Hałdasiowi za udostępnienie części prezentowanych plików źródłowych i elementów slajdów.

Literatura – język C++

- **Jerzy Grebosz**
  Symfonia C ++ Standard, Wydawnictwo E2000, 2006.
- **Delannoy Claude**, Ćwiczenia z języka C++ : programowanie obiektowe, WNT, 1993.

Programowanie obiektowe – założenia I

- **Programowanie obiektowe** – paradygmat programowania w którym używane są „obiekty” i ich interakcje do tworzenia aplikacji.
- **Obiekty** są elementami łączącymi stan (dane) i zachowanie (czyli procedury, nazywane metodami).
- Podejście to różni się od tradycyjnego programowania proceduralnego, gdzie dane i procedury nie są ze sobą bezpośrednio związane. Programowanie obiektowe ma ułatwić pisanie, konserwację i wielokrotne użycie programów lub ich fragmentów.
- Programowanie obiektowe jest oparte na kilku podstawowych technikach:
  - Enkapsulacji,
  - Modularności,
  - Polimorfizmowi,
  - Dziedziczeniu.

Enkapsulacja

- Zmiana stanu (danych) obiektu jest możliwa tylko poprzez jego interfejs.
- Inne obiekty mogą wpływać na jego stan wywołując specjalnie przygotowane metody.
- Dane wewnątrz obiektu są w ten sposób ochronione, a użytkownik obiektu ma do dyspozycji tylko te mechanizmy wpływu na stan obiektu, które udostępnił projektant/programista.
- Niektóre języki programowania obiektowego pozwalają na złamanie lub osłabienie zasady enkapsulacji (w tym język C++).

Modularność

- Możliwość zdefiniowania programu jako zbioru rozdzielnych kooperujących modułów.
- W odniesieniu do programowania strukturalnego, w programowaniu obiektowym „obiekt” może stanowić wydzielony moduł – zawiera on dane oraz funkcje je udostępniające lub modyfikujące.
- W programowaniu strukturalnym funkcje i procedury też są swego rodzaju modułami, jednak dane na których operują są definiowane od nich niezależnie.

Polimorfizm

- Polimorfizm w programowaniu obiektowym to wykazywanie przez metodę różnych form działania w zależności od tego jaki typ obiektu jest wskazywany przez wskaźnik lub referencję (pomijając typ wskaźnika lub referencji).

Dziedziczenie

- W programowaniu obiektowym istnieje możliwość zdefiniowania „podklasy” na podstawie danej klasy.
- Podklasa „dziedziczy” atrybuty i zachowania od rodzica i może wprowadzić własne (np. polegające na uszczegółowieniu funkcjonalności specyficznego przedstawiciela danego typu).
- Najprostsze dziedziczenie odbywa się po jednym „przodku”. W niektórych językach można dziedziczyć po wielu klasach.

Programowanie obiektowe – założenia II

- **Ponadto w metodykach i językach programowania obiektowego często stosuje się pojęcia:**
  - Klasa
  - Obiekt
  - Metoda
  - Przekazywanie wiadomości
  - Abstrakcja

Klasa

- Klasa definiuje w sposób abstrakcyjny rzecz/przedmiot/obiekt. Klasa zawiera cechy/atrybuty przedmiotu (opisane przez dane) oraz zachowanie obiektu (opisane przez funkcje/metody).
- Klasa stanowi wydzielony moduł programu.
- Klasa definiuje możliwości zmiany opisu cech przedmiotu – dostarczając własny interfejs decyduje w jaki sposób mogą być modyfikowane dane.

Obiekt

- Konkretna instancja/reprezentant klasy.
- Klasa definiuje cechy grupy przedmiotów, obiekt reprezentuje pojedynczy, konkretny przedmiot.
- Wartości atrybutów danego obiektu stanowią jego stan.
- Dodatkowo każdy obiekt może się „zachowywać” zgodnie z zestawem zachowań zdefiniowanych w klasie.

Metoda

- Reprezentuje czynności, które może wykonać obiekt.
- W implementacji są to funkcje/procedury, które stanowią interfejs do danych, funkcje/procedury komunikacji z innymi obiektami lub funkcje/procedury reprezentujące pewne umiejętności (realizacje algorytmów).

Przekazywanie wiadomości

- Przekazywanie wiadomości jest metodą wymiany danych powszechnie przyjętą w programowaniu obiektowym
- Komunikacja odbywa się przez wysłanie wiadomości do odbiorcy.

Abstrakcja

- Abstrakcja polega na uproszczeniu złożonej rzeczywistości przez modelowanie klas oraz wybranie odpowiedniego poziomu dziedziczenia odpowiednio do postawionego problemu
- Abstrakcyjny opis rzeczywistości uzyskuje się również przez kompozycję – łączenie kilku klas w jedną całość.

Język C++ - pierwszy program

```
#include<iostream>                          //nowe biblioteki
using namespace std;
void main()
{    cout<<"Witajcie na wykladzie\n\t";     // nowy sposób obsługi We/Wy
     double liczba;                         // zmienna nie musi być
                                            // na początku bloku
     double a=1.2, b=2.6e23, c;
     c=a+b;
     cout<<"Wynik: "<<c<<endl;

     cout<<"Podaj a: ";
     cin>>a;
     cout<<"Podaj b: ";
     cin>>b;
     c=a+b;
     cout<<a<<'+'<<b<<'='<<c<<endl;

     char ch='a';}
```

Przykład:

pierwszy.sln

Obsługa daty – rozwiązanie strukturalne

```
#include<iostream>
using namespace std;
struct Data
{     int dzien;
      int miesiac, rok;
};
void UstalDate(Data& rd, int d, int m, int r)
{     rd.dzien=d;
      rd.miesiac=m;
      rd.rok=r;
}
void WypiszDate(Data afad);
void WpiszDate(Data* pd);

void main()
{     Data dzis;
      UstalDate(dzis,16,12,2006);
      WypiszDate(dzis);
      WypiszDate(dzis);

      Data inna;
      WpiszDate(&inna);
      WypiszDate(inna);
}
```

```
void WypiszDate(Data d)
{     cout<<d.dzien<<'-'<<d.miesiac<<'-
      '<<d.rok<<endl;
      d.dzien++;
}

void WpiszDate(Data* pd)
{
      cout<<"Podaj dzien: ";
      cin>> (*pd).dzien;
      cout<<"Podaj miesiac: ";
      cin>> pd->miesiac;
      cout<<"Podaj rok: ";
      cin>>pd->rok;
}
```

Przykład:

data_struct.sln

Obsługa daty – rozwiązanie obiektowe

```
// dat_obj.h:
#include<iostream>

using namespace std;

class Data
{
//     int tmp;
private:
       int dzien;
       int miesiac;
protected:
       int rok;
public:
       void Wpisz();
       void Wypisz() const;
       void Ustal(int d, int m, int r);
       //Data();
       //virtual ~Data();
};
```

```
//data_obj.cpp
#include "dat_obj1.h"

//Data::Data(){}

//Data::~Data(){}

void Data::Ustal(int d, int m, int r)
{              dzien=d;
               miesiac=m;
               rok=r;}

void Data::Wypisz() const
{      cout<<dzien<<'-
       '<<miesiac<<'-'<<rok<<endl;}

void Data::Wpisz()
{      cout<<"Podaj dzien: ";
       cin>> (*this).dzien;
       cout<<"Podaj miesiac: ";
       cin>> this->miesiac;
       cout<<"Podaj rok: ";
       cin>>this->rok;
}
```

```
//main.cpp
#include "dat_obj1.h"

void main()
{      Data dzis;
       dzis.Ustal(16,12,2006);
       dzis.Wypisz();

       Data inna;
       inna.Wpisz();
//     inna.miesiac=123;
//     inna.rok=2005;
       inna.Wypisz();
}
```

Przykład:

data_obj1.sln

Hermetyzacja/Enkapsulacja

Etykiety dostępu:

- public:**-dostęp dla wszystkich**
- private:**-dostęp tylko dla metod**
- protected:**-tak jak private: (różnice dopiero przy dziedziczeniu)**

Wskaźnik this

- stały wskaźnik do obiektu struktury (klasy)
- niejawny argument funkcji składowej klasy

Język C++ - argumenty domyślne funkcji

```
#include<iostream>
using namespace std;

void Temperatura(double t, int skala=1);
      // skala= 0-st. C, 1-st. F, 2- K

void Temperatura(double t, int skala)
{
 cout<<"t=";

 switch(skala)
 {
 case 0:     cout<<t<<" st. C";   break;
 case 1:     cout<<1.8*t+32<<" st. F";  break;
 case 2:     cout<<t+273.15<<" K";  break;
 default:    cout<<"nieznana skala temperatur";
             break;
 }
 cout<<endl;
}
```

```
void main()
{
 Temperatura(36.6);
 Temperatura(-12.1);
 Temperatura(22);
 Temperatura(-196,2);
 Temperatura(0,1);
 Temperatura(100,1);
}
```

Przykład:

arg_domyslne.sln

Język C++ - przeładowanie funkcji

```
#include<iostream>
using namespace std;

void drukuj(int i)
{ cout<<"To jest zmienna int o wartosci "<<i<<endl;}

void drukuj(double d)
{ cout<<"To jest zmienna double o wartosci
      "<<d<<endl;}

void drukuj(char c)
{ cout<<"To jest znak '" <<c<<"' o kodzie ASCII
      "<<int(c)<<endl;}

void drukuj(int i, char c)
{ cout<<"int: "<<i<<", char: "<<c<<endl;}

void drukuj(char* s)
{ cout<<"To nie jest "<<s<<endl;}
```

```
void main()
{ drukuj("fajny programik");
  drukuj(12,'t');
  drukuj('u');
  drukuj(34.6);
  drukuj(123);}
```

Przykład:

przeladowanie1.sln

Konstruktor

- to metoda nazywająca się tak samo jak klasa
- uruchamia się automatycznie na rzecz nowopowstałego obiektu
- nie konstruuje obiekt klasy tylko inicjalizuje składowe klasy
- nie zwraca żadnej wartości
- może być przeładowywany

Destruktor

- to metoda nazywająca się tak samo jak klasa poprzedzona znakiem ~ (tyldy)
- uruchamia się automatycznie na rzecz obiektu tuż przed jego likwidacją
- nie likwiduje obiektu
- nie posiada żadnych argumentów
- nie zwraca żadnej wartości
- nie może być przeładowywany

Konstruktor/destruktor – pierwszy przykład

```
#include<iostream>
using namespace std;

class Grzeczny
{
public:
 Grzeczny(){ cout<<"Dzien dobry\n";}
 ~Grzeczny(){ cout<<"Do widzenia\n";}
};


void main()
{
 cout<<"Funkcja glowna\n";
}

Grzeczny a;
```

Przykład:

grzeczny.sln

Konstruktor/Destruktor – ciekawszy przykład

```
//stoper.h
#include<time.h>
#include<iostream>

using namespace std;

class Stoper
{
      clock_t timer;
      static int Jest;
public:
      Stoper();
      ~Stoper();

};
```

```
//stoper.cpp
#include "Stoper.h"

Stoper t1;
int Stoper::Jest=0;//<-
Stoper::Stoper()
{     if(!Jest) timer=clock();
      ++Jest;
}
Stoper::~Stoper()
{     --Jest;
      if(!Jest)
      {
       timer=clock()-timer;
       cout<<"Czas dzialania programu wynosi:”
            <<double(timer)/CLOCKS_PER_SEC
            <<" s\n"<<flush;
       cout<<"Do zrobaczenia"<<endl;
      }}
```

```
#include <iostream>

using namespace std;

void main()
{
 double a=1.1,b;
 int i=1000000000;
 cout << "Start programu..."
        <<endl;
 while(i>0)
 {
  b=a/2.3;
  --i;
 }
}
```

Przykład:

stoper.sln

Dziedziczenie

- Dziedziczenie to technika pozwalająca na definiowanie nowej klasy przy wykorzystaniu klasy już wcześniej istniejącej
- W klasie pochodnej możemy:
  - Zdefiniować dodatkowe dane składowe
  - Zdefiniować dodatkowe funkcje składowe
  - Zdefiniować składnik (najczęściej funkcję składową), który już istnieje w klasie podstawowej.

Dziedziczenie - przykład

```
//skarbonka.h
class Skarbonka
{ protected:
                int aktywa;
 public:
      int Wyjmij_gotowke(unsigned int ile);
      int Ile_w_skarbonce();
      void Wrzuc_co_laska (int ile);
      Skarbonka();};
```

```
//Skarbonka.cpp
#include "Skarbonka.h"
Skarbonka::Skarbonka()  {aktywa=0;}
void Skarbonka::Wrzuc_co_laska(int ile)
{ aktywa+=ile;}
int Skarbonka::Ile_w_skarbonce()  {return aktywa;}
int Skarbonka::Wyjmij_gotowke(unsigned int ile)
{     if(unsigned(aktywa)>ile) { aktywa-=ile;}
      else     {ile=aktywa;
                aktywa=0;}
      return ile; }
```

```
// konto.h:
#include "Skarbonka.h"
#include<iostream>
using namespace std;
class Konto : public Skarbonka
{ protected:
      int PIN;
 public:
      void Bankomat();
      Konto();};
```

```
//konto.cpp
#include "Konto.h"
Konto::Konto() { PIN=3421;}
void Konto::Bankomat()
{ cout<<"Podaj PIN: ";
  int pin;
  cin>>pin;
 if(pin==PIN)
 {  cout<<"Podaj kwote: ";
    int kwota;
    cin>>kwota;
    if(kwota<Ile_w_skarbonce())
  {  Wyjmij_gotowke(kwota);  }}}
```

Dziedziczenie - przykład

Przykład:

swinka_konto.sln

```
//main_dostep.cpp

void main()
{
 Skarbonka swinka;

 swinka.Wrzuc_co_laska(123);
 //swinka.aktywa=100;
 swinka.Wyjmij_gotowke(100);
 cout << swinka.Ile_w_skarbonce() << endl;

 Konto ROR;
 //ROR.aktywa=100;
 cout << ROR.Ile_w_skarbonce() << endl;
 ROR.Wrzuc_co_laska(124);
 cout << ROR.Ile_w_skarbonce() << endl;
 ROR.Bankomat();
 ROR.Wrzuc_co_laska(124);
 cout << ROR.Ile_w_skarbonce() << endl;
}
```

Komunikacja/Modularność - przykład

```
#include <iostream>
#include <time.h>
using namespace std;
class a
{
protected:
	int dana_a;
public:
	int  zwroc_dana(void)
	{		return dana_a;			};
	void zmieniaj(void)
	{		dana_a=(rand()%100);
			cout<<"Nowa dana_a:"<<dana_a<<endl;};
	a(){dana_a=100;
			cout<<"inicjalizacja a:"<<dana_a<<endl;
	   srand((unsigned)time(NULL));};
};
class b
{
protected:
	int dana_b;
public:
	void wpisz(int dana)
	{		dana_b=dana;			}
	void wypisz(void)
	{		cout<< dana_b<<endl;	}
	void odbierz(a obj)
	{		int tmp;
			tmp=obj.zwroc_dana();
			wpisz(tmp);};
	b(){	dana_b=0;	};
};
```

```
void main(void)
{	a obj1;
	b obj2;

	obj2.wypisz();

	int tmp;
	tmp=obj1.zwroc_dana();
	obj2.wpisz(tmp);

	obj2.wypisz();
	obj1.zmieniaj();
	obj2.odbierz(obj1);
	obj2.wypisz();
}
```

Przykład: komunikacja.sln

Agregacja - przykład

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

class a
{
protected:
	int dana_a;
public:
	void wyslij(int &dana){dana=dana_a;};
	void zmieniaj(void)
	{ dana_a=(rand()%100); };
	a(){dana_a=100;
	    srand((unsigned)time(NULL));};
};

class b
{
protected:
	int dana_b;
public:
	void odbierz(a liczba){liczba.wyslij(dana_b);
	printf("%d\n",dana_b);};
};
```

```
// Agregacja klas
class c
{
public:
	a a1;
	b b1;
	void komunikacja(void)
	{
		while(1)
		{// Modularność:
			a1.zmieniaj();
			b1.odbierz(a1);
			Sleep(500);
		}
	};
};


void main(void)
{
c c1;
c1.komunikacja();
}
```

Przykład: agregacja.sln