

# Programowanie obiektowe

## na przykładzie języka C++

Celem tego wykładu jest nauka programowania obiektowego, a nie nauka języka C++. Ale bez paniki: języka C++ nauczymy się “przy okazji”.

Piotr Białas

pok 443, tel 5571

pon. 11<sup>00</sup>-12<sup>00</sup>, śr. 10<sup>00</sup>-11<sup>00</sup>

pbialas@th.if.uj.edu.pl

# Wykład 1

- Wstęp teoretyczno – filozoficzny
  - Proces konstrukcji oprogramowania
  - Składniki podejścia obiektowego
  - Co to jest obiekt i klasa
- C++
  - Definiowanie klas i obiektów
  - Cykl życia obiektu (tworzenie i niszczenie)

# Proces konstrukcji oprogramowania

- Analiza
  - Odpowiedź na pytanie "co?"
- Projektowanie
  - Odpowiedź na pytanie "jak?"
- Kodowanie (programowanie)
  - Implementacja powyższej odpowiedzi

Podejście obiektowe można (a właściwie należy) stosować na każdym z tych etapów.

# Programowanie obiektowe

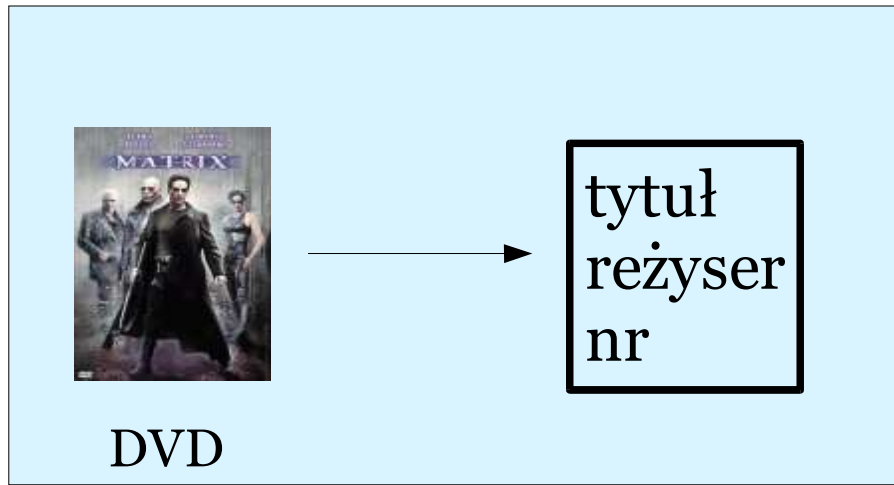
“Heaven is not a place, heaven is a feeling.”

Programowanie obiektowe to nie jest programowanie w języku obiektowym, to pewien sposób myślenia.

“Programowanie obiektowe to metoda implementacji w której programy są zorganizowane jako zbiór współpracujących obiektów, każdy z których jest instancją pewnej klasy i które to klasy połączone są ze sobą związkami hierarchii dziedziczenia.”

Grady Booch, “Object Oriented Analysis and Design”

# Przerwa semantyczna



C++

C

Fortran

assembler

mikrokod

bramki logiczne

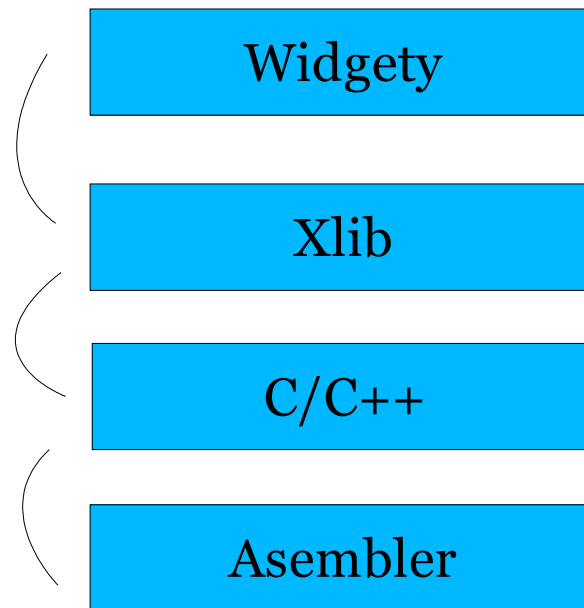
Wyabstrachowane przez nas pojęcia  
musimy zareprezentować w  
komputerze

# Zmniejszanie przerwy semantycznej

- Przerwę semantyczną możemy zmniejszać
  - Od góry (dostosowywując się do komputera)
    - Definiując prostsze abstrakcje tak aby można je było łatwo implementować
  - Od dołu (dostosowywując komputer do nas)
    - używając języków programowania wyższego poziomu
    - używając architektury warstwowej

# Architektura warstwowa

Każda warstwa reprezentuje wyższy poziom abstrakcji



# Języki programowania

- Każdy język programowania posiada zestaw abstrakcji które są w nim łatwe do wyrażenia
- Fortran (Formula Translator)
  - wzory matematyczne
  - podprocedury (dekompozycja funkcyjna)
- Algol, C
  - struktury
- Simula, C++, Java
  - obiekty
- Lisp (LISt Procesor)
  - listy



# Języki programowania

Teoretycznie wszystkie języki są równoważne (Turing) ale każdy język programowania wnosi ze sobą zespół naturalnych dla niego pojęć służących do opisu (dekompozycji) problemu

# Składniki podejścia obiektowego

---

- Abstrakcja
- Enkapsulacja
- Hierarchia
- System typów

Te pojęcia istniały przed pojawieniem się języków obiektowych, dopiero języki obiektowe dały im praktyczne wsparcie

# Abstrakcja

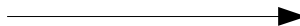
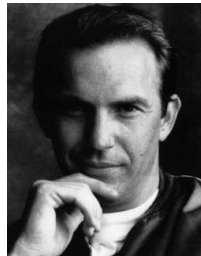
Umiejętność wyodrębnienia cech istotnych dla **danego** problemu

DVD



Tytuł  
Reżyser  
nr

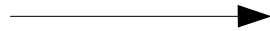
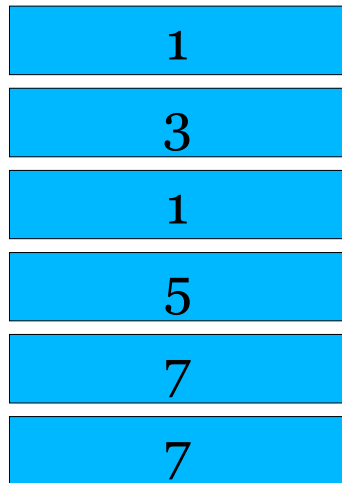
Wypożyczający



Imię  
Nazwisko  
Adres  
Data ur.  
nr Klienta

# Abstrakcja

Stos



```
int pop()  
push(int)  
int jestPusty()
```

Stos to “coś” na czym można wykonywać operacje pop i push

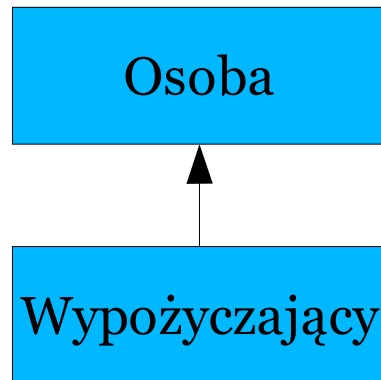
# Enkapsulacja

---

- Użyteczna abstrakcja w praktyce musi być niezależna od swojej implementacji. Osiągamy to poprzez
- **Rozdzielenie interfejsu od implementacji**
- Jest to jedna z podstawowych zasad projektowych nie tylko w podejściu obiektowym
- ale języki obiektowe dają nam wygodne mechanizmy do tego celu

# Hierachia

- 
- 



Hierarchia “część – całość” możliwa była do prostego zaimplementowania w nieobiekтовых językach programowania. Dziedziczenie przyszło wraz z językami obiekowymi

# Dziedziczenie i interfejsy

- Są dwa główne powody dziedziczenia:
  - Dla interfejsu
    - Klasa dziedzicząca może być użyta wszędzie tam gdzie klasa dziedziczona
  - Dla implementacji
    - Klasa dziedzicząca wykorzystuje i/lub rozszerza działanie klasy dziedziczonej
- Można używać obu na raz

# System typów

- W językach posiadających system typów każda zmienna ma określony typ
- Jeśli typ zmiennej jest znany podczas kompilacji to kompilator może nam zabronić wykonywania niekonsystentnych operacji
- Jeśli typ zmiennej określany jest podczas wykonania to wynik wywołania operacji na tej zmiennej może zależeć od jej typu (polimorfizm).



# System typów w C++

- W C++ mamy do czynienia z podejściem mieszanym: w zasadzie każda zmienna ma typ określony podczas kompilacji ale można do niej podstawiać obiekty innych typów będących pochodnymi jej typu (połączonymi hierarchią dziedziczenia).

# Obiekt i klasa

- Obiekt to coś co posiada
  - Stan
    - Zmienne/attributy
  - Zachowanie
    - Metody/funkcje
  - Tożsamość
    - Osobny obszar pamięci
- Klasa określa strukturę obiektu. Jest to “matryca” do produkcji obiektów

# Stan

---

- Stan jest określony przez listę posiadanych atrybutów (własności), zwykle statyczną i zdefiniowaną w klasie do której należy dany obiekt
- oraz wartości tych atrybutów które zmieniają się w trakcie życia obiektu

# Zachowanie

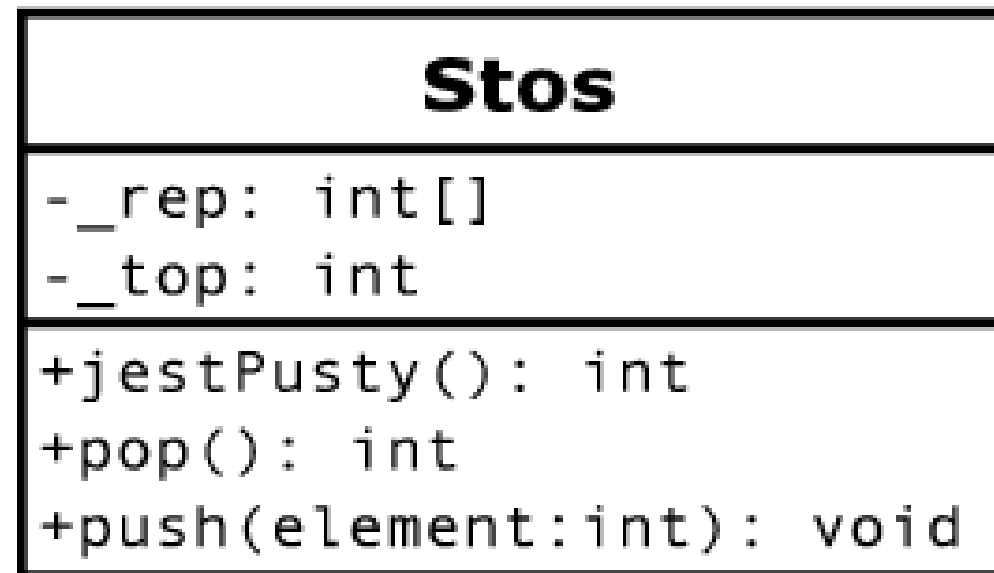
---

- Zachowanie określone jest przez listę metod (funkcji) które możemy wywołać na danym obiekcie
- Wynik wywołania metody zależy od stanu obiektu
- Stan obiektu może zostać zmieniony przez wywołanie metody

# Tożsamość

- Każdy obiekt jest różny od innego obiektu, nawet jeśli posiada identyczny stan
- Nie należy mylić obiektu z jego nazwą (zmienną)
- Aby zachować tożsamość obiektu należy zwracać baczną uwagę na operacje przypisanie i kopiowania

# Unified Modeling Language



# Klasa w C++ - definicja/deklaracja

```
#ifndef _stosint_  
#define _stosint_
```

stosint.h

```
#include<stdlib.h>  
using namespace std;
```

```
static const size_t STACK_SIZE = 100;
```

```
class StosInt {  
private:  
    int _rep[STACK_SIZE];  
    size_t _top;
```

```
public:  
    StosInt():_top(0) {};
```

```
    int jestPusty();  
    int pop();  
    void push(int);
```

```
};
```

```
#endif  
Programowanie obiektowe wykład 1
```

# Klasa C++ - implementacja

```
#include "stosint.h"
```

```
stosint.cpp
```

```
int StosInt::jestPusty() {  
    return !_top;  
}
```

```
int StosInt::pop() {  
    return _rep[--_top];  
}
```

```
void StosInt::push(int element) {  
    _rep[_top++] = element;  
}
```



# Klasa C++ - użycie

```
#include<iostream>
using namespace std;
#include"stosint.h"
main()
{
    StosInt stosik;

    stosik.push(27);stosik.push(34);stosik.push(78);

    while(!stosik.jestPusty()) {
        cerr<<stosik.pop()<<endl;
    }
}
```

stosik.cpp

# C++ - kompilacja i wykonanie (Linux)

```
> g++ -o stosik stosik.cpp stosint.cpp
```

```
> stosik
```

```
78
```

```
34
```

```
27
```

# C++ - enkapsulacja

```
#include<iostream>

using namespace std;

#include"stosint.h"

main()
{
    StosInt stosik;

    if(stosik._top > 0)
    {
        cerr<<"stos niepusty"<<endl;
    }

}
```

# C++ - enkapsulacja

```
stosint.h: In function `int main()':  
stosint.h:13: error: `size_t StosInt::_top' is  
private  
zly.cpp:11: error: within this context
```

# Tworzenie obiektów

- Obiekt jest tworzony za pomocą konstruktora. Jeśli nie zdefiniujemy własnego konstruktora to kompilator wygeneruje standardowy konstruktor który wywoła po kolei standardowe konstruktory wszystkich składowych obiektu
- Możemy zdefiniować kilka konstruktorów. O tym który konstruktor zostanie wykorzystany zadecydują przekazane mu parametry (przeładowanie)

# Niszczenie obiektów

- Przed zniszczeniem obiektu wywoływany jest destruktory, podobnie jak w przypadku konstruktora jeśli nie zdefiniujemy własnego destruktora (może być tylko jeden) to wywoływany jest destruktory standardowy który po kolei niszczy składowe obiektu i zwalnia przydzieloną mu pamięć
- UWAGA! Destruktor standardowy nie zwalnia pamięci przydzielanej dynamicznie. Konieczność zwolnienia przydzielonej dynamicznie pamięci to najczęstszy powód pisania własnego destruktora

# Stos dynamiczny

```
#ifndef _stosdyn_
#define _stosdyn_

#include<stdlib.h>

using namespace std;

class DynamicznyStosInt {
private:
    int *_rep;
    size_t _top;

public:
    DynamicznyStosInt(size_t n):_top(0) {_rep= new int[n]; };

    int jestPusty();
    int pop();
    void push(int);

    ~DynamicznyStosInt() {delete [] _rep;}

};

#endif
```

31 Programowanie obiektowe wykład 1

# Stos dynamiczny

```
#include<iostream>
#include"stosdyn.h"

using namespace std;

main()
{
    DynamicznyStosInt stosik(1000);
    DynamicznyStosInt stosik; //niepoprawne

    for(int i = 0;i<2000000;i++)
    {
        DynamicznyStosInt *s= new DynamicznyStosInt(1000);
        s->push(1);
        delete s;
    }
}
```