

# Programowanie obiektowe

## Wykład 1

Marcin Młotkowski

23 lutego 2017

# Plan wykładu

- 1 Sprawy organizacyjne
  - Opis wykładu
  - Zaliczenie i egzamin
  - Literatura
- 2 Style programowania
- 3 Dlaczego objekty
  - Modelowanie obiektowe
  - Programowanie obiektowe
    - Abstrakcja
    - Hermetyzacja
    - Dziedziczenie
    - Polimorfizm
- 4 Zakończenie

## Strona i konsultacje

Konsultacje

pok. 303

Strona wykładu

<http://www.ii.uni.wroc.pl/~marcinm/dyd/obiekty>

## Cele wykładu

- Poznanie pojęć związanych z programowaniem obiekowym
- Nabycie sprawności w posługiwaniu się mechanizmami obiekowymi
- Zdobywanie umiejętności tworzenia dobrego oprogramowania za pomocą mechanizmów obiektywnych

# Plan wykładu

## Programowanie obiektowe

Java, C#, Ruby

# Plan wykładu

## Programowanie obiektowe

Java, C#, Ruby

Projektowanie i analiza obiektowa

# Plan wykładu

## Programowanie obiektowe

Java, C#, Ruby

Projektowanie i analiza obiektowa

Przegląd innych zagadnień związanych z programowaniem  
obiektywnym

# Pracownia

## Pracownia

Listy zadań, z każdej listy będą dwa zadania do wyboru za 4 pkt, z wyjątkiem 1. listy. Zadania są do oddania na pracowni.



# Pracownia

## Pracownia

Listy zadań, z każdej listy będą dwa zadania do wyboru za 4 pkt, z wyjątkiem 1. listy. Zadania są do oddania na pracowni.

## Projekt

Samodzielnie zaprojektowanie i implementacja wybranego zadania, przedstawienie projektu na pracowni.

## Zaliczenie pracowni

Na ocenę z pracowni składają się punkty za:

- zaprogramowane zadania z list;
- punkty uzyskane za projekt.

## Zaliczenie pracowni

Na ocenę z pracowni składają się punkty za:

- zaprogramowane zadania z list;
- punkty uzyskane za projekt.

### Warunek zdobycia zaliczenia

- zdobycie co najmniej połowy z maksimum do zdobycia punktów (pracownia i projekt);
- zaliczenie projektu na przynajmniej połowę maksymalnej liczby punktów.

## Ocena zadań z listy

Progamy są oceniane na pracowni przez prowadzącego.

## Ocena zadań z listy

Programy są oceniane na pracowni przez prowadzącego.

Przed oddaniem należy program unieść w systemie SKOS  
<https://skos.ii.uni.wroc.pl/> do środy do 8:00.

## Ocena zadań z listy

Programy są oceniane na pracowni przez prowadzącego.

Przed oddaniem należy program unieść w systemie SKOS  
<https://skos.ii.uni.wroc.pl/> do środy do 8:00.

**Zdobyte punkty**

Zdobyte punkty będą prezentowane w systemie USOSWeb.

# Egzamin

Ocena z egzaminu

jest przepisywana z zaliczenia

## Źródła wiedzy

- Dokumentacja firmowa i Internet
- Książki
- Wykład ;-)







# Bibliografia, języki programowania

C#, Java i Ruby

Dokumentacja firmowa, internet

## Bibliografia (1)

-  M. Weisfeld. [Myślenie obiektowe w programowaniu](#)
-  H. Ledgard. [Mała księga programowania obiektowego](#)
-  B. Meyer. [Programowanie zorientowane obiektowo](#)
-  Brett D. McLaughlin et. al. [Object-Oriented Analysis & Design](#)

## Bibliografia (2)



E. Gamma et. al. Wzorce projektowe



I. Graham. Metody obiektowe w teorii i praktyce



E. Yourdon



G. Booch

## Bibliografia (3)

### Inne języki



B. Stroustrup. Język C++



G. Shlossnagle. PHP Zaawansowane programowanie

### Teoria języków programowania



M. Abadi, L. Cardelli. Theory of Objects



K. Bruce. Foundations of Object-Oriented Languages:  
Types and Semantics

# Plan wykładu

- 1 Sprawy organizacyjne
  - Opis wykładu
  - Zaliczenie i egzamin
  - Literatura
- 2 Style programowania
- 3 Dlaczego obiekty
  - Modelowanie obiektowe
  - Programowanie obiektowe
    - Abstrakcja
    - Hermetyzacja
    - Dziedziczenie
    - Polimorfizm
- 4 Zakończenie

## Programowanie funkcjonalne (funkcyjne)

### Niektóre cechy

- program jest reprezentowany przez funkcję (funkcje);
- redukcja efektów ubocznych;
- przykłady: LISP, ML czy Haskell.

## Przykład programu funkcyjnego

### Definicja silni

#### definicja matematyczna

$$\text{silnia}(n) = \begin{cases} 1 & \text{gdyn} = 0 \\ n * \text{silnia}(n - 1) & \text{wpp} \end{cases}$$

#### Implementacja w Ocaml'u

```
let rec silnia n =  
  if n=0 then 1  
  else n*silnia(n-1);;
```

# Programowanie deklaratywne

## Cechy

- programista określa *jaki wynik go interesuje*; nie podaje *jak to zrobić*;
- algorytm wyszukiwania jest częścią języka;
- przykład: Prolog, SQL.



## Programowanie deklaratywne

### Cechy

- programista określa *jaki wynik go interesuje*; nie podaje *jak to zrobić*;
- algorytm wyszukiwania jest częścią języka;
- przykład: Prolog, SQL.

```
member(X, [X|_]).  
member(X, [_|Y]) :-  
    member(X, Y).
```

## Programowanie deklaratywne

### Cechy

- programista określa *jaki wynik go interesuje*; nie podaje *jak to zrobić*;
- algorytm wyszukiwania jest częścią języka;
- przykład: Prolog, SQL.

```
member(X, [X|_]).  
member(X, [_|Y]) :-  
    member(X, Y).
```

```
SELECT Nr_albumu FROM Studenci  
WHERE objekty_punkty > 30;
```

# Programowanie strukturalne

Algorytmy + struktury danych = programy

## Cechy

- program jest dzielony na bloki z jednym punktem wejścia;
- zakaz używania instrukcji skoku;
- przykłady języków: Pascal, Ada, Modula.

# Programowanie obiektowe

## Cechy

- programy są implementowane za pomocą obiektów;
- przykłady: SIMULA, SmallTalk, Java, C#.

# Plan wykładu

- 1 Sprawy organizacyjne
  - Opis wykładu
  - Zaliczenie i egzamin
  - Literatura
- 2 Style programowania
- 3 **Dlaczego obiekty**
  - **Modelowanie obiektowe**
  - **Programowanie obiektowe**
    - Abstrakcja
    - Hermetyzacja
    - Dziedziczenie
    - Polimorfizm
- 4 Zakończenie

## Pisanie programów na zamówienie

### Specyfikacja programu

„Napisz mi program, w którym bym notował elektronicznie różne fakty związane z moim akwarium, na przykład liczbę i gatunki ryb, temperaturę, częstotliwość karmienia.”

## Pisanie programów na zamówienie

### Specyfikacja programu

„Napisz mi program, w którym bym notował elektronicznie różne fakty związane z moim akwarium, na przykład liczbę i gatunki ryb, temperaturę, częstotliwość karmienia.”

## Pisanie programów na zamówienie

### Specyfikacja programu

„Napisz mi program, w którym bym **notował** elektronicznie różne **fakty** związane z moim **akwarium**, na przykład liczbę i **gatunki ryb**, **temperaturę**, częstotliwość **karmienia**.”



## Analiza specyfikacji

- Implementacja obiektów świata rzeczywistego, np. osób, procesów, dokumentów, zdarzeń.
- Niewiele informacji o tym, co należy zrobić z danymi.
- Specyfikacja jest *bardzo nieformalna*, dalsze uszczegółowienie może sporo pozmieniać.
- Na pewno coś się zmieni (potrzeby użytkownika, prawo, system operacyjny etc).

# Modelowanie danych

Naturalne jest najpierw zdefiniowanie pojęcia (danej), a potem określenie operacji, jakie będziemy na tym pojęciu wykonywali

# Obiekty

## Obiekt rzeczywisty

**Obiekt:** rzeczywiste pojęcie (osoba, przedmiot, zdarzenie etc) posiadające pewne właściwości (rozmiar, czas, kolor).

## Reprezentacja obiektu

**Obiekt:** reprezentacja rzeczywistego pojęcia w programie. Obiekty mają cechy reprezentowane za pomocą pól, zwane stanem obiektu, oraz zbiór operacji.

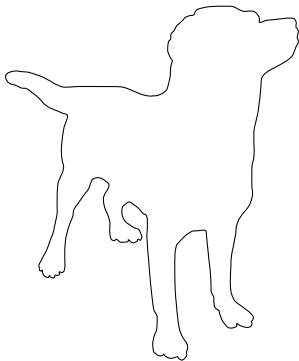
## Zalety programowania obiektowego

Implementacja obiektowa w naturalny sposób reprezentuje objekty rzeczywiste.

## Zalety programowania obiektowego

Implementacja obiektowa w naturalny sposób reprezentuje obiekty rzeczywiste.

pojęcie rzeczywiste	⇒	klasa
cechy rzeczywistego obiektu	⇒	pola obiektu
operacje na obiekcie	⇒	metody obiektu



```
class Dog {
```



```
class Dog {  
    Color color;  
}
```



```
class Dog {  
    Color color;  
    public void szczekanie() {  
        System.out.println("Hau hau");  
    }  
}
```



# Cechy programowania obiektowego

- abstrakcja
- hermetyzacja (enkapsulacja)
- dziedziczenie
- polimorfizm

# Abstrakcja

## Nieformalna definicja abstrakcji

Definiujemy co należy zrobić, nie definiujemy jak.

# Abstrakcja

## Nieformalna definicja abstrakcji

Definiujemy co należy zrobić, nie definiujemy jak.

## Definicja stosu liczb całkowitych: operacje

```
void push(int element)
```

```
int pop()
```

```
boolean empty()
```

## Własności listy

$$\forall x : \{lista.push(x); y = lista.pop()\} \Rightarrow x = y$$

## Pojęcie hermetyzacji (enkapsulacji)

Ukrywanie "pomocniczych" elementów obiektu (pól i metod) i jawne udostępnianie wskazanych funkcjonalności: interfejs.

Zalety hermetyzacji:

- odporność na zepsucie;
- łatwa zmiana implementacji.

## Pojęcie hermetyzacji (enkapsulacji)

Ukrywanie "pomocniczych" elementów obiektu (pól i metod) i jawne udostępnianie wskazanych funkcjonalności: interfejs.

Zalety hermetyzacji:

- odporność na zepsucie;
- łatwa zmiana implementacji.

### Przykład

Implementacja listy jednokierunkowej

# Strukturalna implementacja listy

```
struct listElem {  
    int elem;  
    struct listElem *lista;  
}
```

```
void  
wstaw(struct listElem *lista, int elem);
```

```
int  
szukaj(struct listElem *lista, int elem);
```

# Strukturalna implementacja listy

```
struct listElem {  
    int elem;  
    struct listElem *lista;  
}
```

```
void  
wstaw(struct listElem *lista , int elem);
```

```
int  
szukaj(struct listElem *lista , int elem);
```

## Komentarz

- Dane i operacje na danych są rozdzielone
- Jawna implementacja struktury danych
- Niebezpieczeństwo naruszenia integralności danych przez "obcy" kod.

# Schemat implementacji obiektowej

```
class Lista {  
    int arr[] = new int[1000];  
    int pos = 0;  
    public void wstaw(int elem) {  
        ...  
    }  
    public boolean szukaj(int elem) {  
        ...  
    }  
}
```



# Schemat implementacji obiektowej

```
class Lista {  
    int arr[] = new int[1000];  
    int pos = 0;  
    public void wstaw(int elem) {  
        ...  
    }  
    public boolean szukaj(int elem) {  
        ...  
    }  
}
```

Próba zepsucia

lista.pos = -3

# Implementacja obiektowa, wersja 1.

```
class Lista {  
    int arr[] = new int[1000];  
    int pos = 0;  
  
    public void wstaw(int elem) {  
        arr[pos] = elem;  
        pos++;  
    }  
  
    public boolean szukaj(int elem) {  
        for (int i = 0; i < pos; i++)  
            if (arr[i] == elem) return true;  
        return false;  
    }  
}
```

## Implementacja obiektowa, wersja 2.

```
class Lista {
    Lista next;
    int store;
    public void wstaw(int elem) {
        Lista cursor = this;
        while (cursor.next != null) cursor = cursor.next;
        cursor.next = new Lista();
        cursor.next.store = elem;
    }
    public boolean szukaj(int elem) {
        Lista cursor = this;
        while (cursor != null) {
            if (cursor.store == elem) return true;
            cursor = cursor.next;
        }
        return false;
    }
}
```

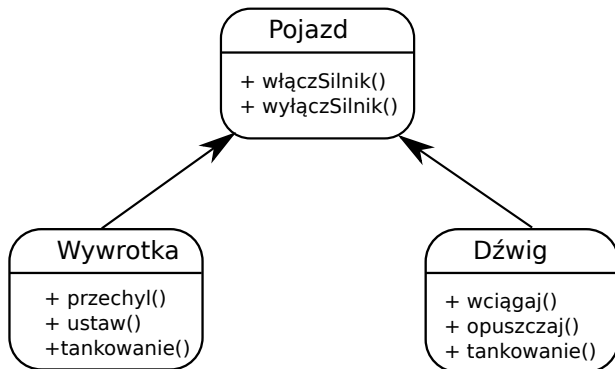
## Cechy implementacji obiektowej

- Struktura danych i operacje są w jednym miejscu;
- dostęp do implementacji tylko poprzez wyspecyfikowane metody;
- stała lista operacji (wstaw i szukaj);
- zmienna implementacja, niewidoczna "z zewnątrz".

# Definicja dziedziczenia

**Dziedziczenie:** utworzenie klasy na podstawie innej, już istniejącej klasy.

## Przykład dziedziczenia



## Zalety dziedziczenia

- uporządkowanie podobnych pojęć;
- wielokrotne wykorzystanie tego samego kodu.

# Co to jest polimorfizm

## Opis

Z polimorfizmem mamy do czynienia gdy jedna nazwa (np. nazwa metody) może oznaczać różne (w sensie implementacji) faktyczne metody.



## Przykład polimorfizmu

```
1 Pojazd p = new Wywrotka();  
2 p.tankowanie();  
3 p = new Dzwig();  
4 p.tankowanie();
```

## Przykład polimorfizmu

```
1  
2 p.tankowanie ();  
3  
4 p.tankowanie ();
```

## Inny polimorfizm: +, \*

### Liczby typu `int`

$$2 + 2$$

$$3 * 4$$

$$x * (y + z)$$

### Liczby typu `float`

$$2.71 + 3.0$$

$$2 * 3.14$$

$$x * (y + z)$$

## Inny polimorfizm: +, \*

### Liczby typu `int`

$$2 + 2$$

$$3 * 4$$

$$x * (y + z)$$

### Liczby typu `float`

$$2.71 + 3.0$$

$$2 * 3.14$$

$$x * (y + z)$$

### Macierze

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad x * (y + z)$$

# Plan wykładu

- 1 Sprawy organizacyjne
  - Opis wykładu
  - Zaliczenie i egzamin
  - Literatura
- 2 Style programowania
- 3 Dlaczego obiekty
  - Modelowanie obiektowe
  - Programowanie obiektowe
    - Abstrakcja
    - Hermetyzacja
    - Dziedziczenie
    - Polimorfizm
- 4 Zakończenie

# Języki obiektowe

**Java**, SmallTalk, Ocaml, C++, Python, Self, C#, Simula, Eiffel,  
CommonLisp, ...

## Programowanie obiektowe — uwagi

### Uwaga 1.

Można programować *w stylu obiektowym* w językach nieobiektowych.

## Programowanie obiektowe — uwagi

### Uwaga 1.

Można programować *w stylu obiektowym* w językach nieobiektowych.

### Uwaga 2.

Używanie języka obiektowego nie oznacza że się programuje obiektowo.