Programowanie Obiektowe i C++

Marcin Benke

2.10.2006

Dzisiaj

- Co umiemy
- Paradygmaty programowania
- Co będzie na wykładach
- Zasady zaliczania
- Programowanie obiektowe

Co umiemy

- Programowałem w C++
- Programowałem w języku obiektowym
- Programowałem w C
- Programowałem w innym języku
- Nigdy nie programowałem

Paradygmaty programowania

Ewolucja języków programowania od bliskich maszynie do bliskich rozwiązywanym problemom:

- Imperatywne
  - proceduralne — "książka kucharska"
  - obiektowe — interakcje obiektów
- Deklaratywne
  - funkcyjne — funkcje i wartości
  - logiczne — relacje, spełnianie formuł
- Specyficzne dla dziedziny (*domain specific*)

Przykłady języków

- proceduralne: Algol, Pascal, C,...
- obiektowe: Smalltalk, C++, Java, Python, Ruby,...
- funkcyjne: Lisp, ML, Haskell,...
- logiczne: Prolog, Mercury,...

W praktyce większość języków łączy różne paradygmaty.

Co będzie na wykładach

- Programowanie proceduralne w C++
- Programowanie obiektowe w C++ (klasy, dziedziczenie, metody wirtualne, szablony, ...),
- Programowanie ekstremalne
- Interakcja z użytkownikiem
- Inne języki obiektowe (jeśli starczy czasu)

Dlaczego C++?

- bardzo popularny,
- dostępny w wielu implementacjach,
- efektywne implementacje
- łatwo dostępna literatura

Zastrzeżenia

- C++ nie jest jedynym językiem obiektowym,
- C++ nie jest najlepszym językiem obiektowym,
- C++ jest trudnym językiem, z bardzo rozbudowaną składnią i subtelną semantyką
- w C++ można pisać programy nie mające nic wspólnego z programowaniem obiektowym,

"W każdym języku da się programować w Fortranie"

Zaliczenie ćwiczeń

- obecności,
- kolokwium w połowie semestru,
- program zaliczeniowy.

Programowanie obiektowe

- Wszyscy o nim mówią, wszyscy go używają i nikt nie wie co to jest.
- Podejście obiektowe wykracza daleko poza programowanie (ogólnie: opis skomplikowanych systemów).
- Inny sposób widzenia świata
  - Agenci, do których wysyła się komunikaty (zlecenia wykonania pewnych zadań)
  - Metody realizacji zleceń są ukryte (przykłady: poczta, WWW)

.

Nowy wspaniały świat

- Więcej niż jedynie dodanie do języka programowania kilku nowych cech,
- inny sposób myślenia o projektowaniu i tworzeniu programów.
- Na programowanie obiektowe można patrzeć jako na symulowanie rozważanego świata.
- W programowaniu obiektowym mamy do czynienia z zupełnie innym modelem obliczeń:
- zamiast komórek pamięci i ciągu instrukcji mamy obiekty (agentów) komunikaty i zobowiązania.

Obiekty i klasy

- Obiekt:
  - stan (atrybuty),
  - zachowanie (operacje, metody).
- Każdy obiekt jest egzemplarzem pewnej klasy.
- Zachowanie obiektu jest określone w jego klasie.
- Z każdym obiektem jest związany pewien zbiór zobowiązań — protokół
- Inaczej: protokół określa na jakie komunikaty obiekt odpowiada (jakie zlecenia wykonuje).

Komunikaty

- Zachowanie obiektu można zaobserwować wysyłając do niego komunikat;
- w odpowiedzi obiekt wykona swoją metodę związaną z tym komunikatem;
- To jakie akcje zostaną wykonane zależy od obiektu — obiekt innej klasy może wykonać w odpowiedzi na ten sam komunikat zupełnie inne akcje (polimorfizm).

Komunikaty i metody

- Wysłanie komunikatu jest podobne do wywołania procedury.
- Istotna różnica: to jakie akcje zostaną wykonane zależy od odbiorcy komunikatu.
- Wiązanie nazwy komunikatu z metodą odbywa się w czasie wykonania, a nie kompilacji (metody wirtualne, wczesne i późne wiązanie metod).

Delegacja i komponenty

- Staramy się co tylko się da zrzucić na innych (agentów), a nie robić wszystko samemu.
- Sprzyja to tworzeniu komponentów nadających się do ponownego wykorzystania w inych programach (reuse)
- Projektowanie sterowane zobowiązaniami (*Responsibility Driven Design*):
  - delegowanie zadań daje agentom większą samodzielność,
  - komponenty stają się mniej zależne od siebie,
  - to z kolei ułatwia ich ponowne wykorzystanie.

Ewolucja

- Kolejny etap ewolucji mechanizmów abstrakcji:
  - procedury
  - bloki
  - moduły
  - ATD
  - programowanie obiektowe

Własności OOP

Alan Kay (1993):

- Wszystko jest obiektem.
- Obliczenie realizują obiekty przesyłając między sobą komunikaty.
- Obiekt ma swoją pamięć zawierającą inne obiekty (odwołania do nich).
- Każdy obiekt jest egzemplarzem klasy.
- Klasa jest wzorcem zachowania obiektu.
- Klasy są zorganizowane w hierarchię dziedziczenia.

Dziedziczenie

- Jeden z fundamentów podejścia obiektowego.
- Klasy obiektów można kojarzyć w hierarchie klas (prowadzi to do drzew lub grafów dziedziczenia).
- Atrubuty i zachowanie klas-przodków są dostępne w klasach potomków (pośrednio lub bezpośrednio).
- Potomek może mieć odmienne zachowanie niż przodek.
- Nadklasy i podklasy (klasy bazowe i pochodne).
- Zasada podstawialności: zawsze powinno być możliwe podstawienie obiektów podklas w miejsce obiektów nadklas.

Przykład dziedziczenia — Point

Rozważmy następującą klasę:

```
class Point {
attributes:
  int x, y
methods:
  setX(int newX)
  getX()
  setY(int newY)
  getY()
}
```

(Uwaga: to jest pseudokod, jeszcze nie w pełni C++.)

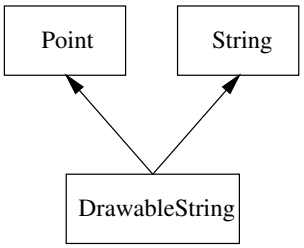
Przykład dziedziczenia — Circle

```
class Circle inherits from Point {
attributes:
  int radius
methods:
  setRadius(int newRadius)
  getRadius()
}

Circle acircle
acircle.setX(1)        /* Dziedziczone z Point */
acircle.setY(2)
acircle.setRadius(3)   /* Dodane przez Circle */
```

Wielodziedziczenie

- Wielodziedziczenie *(multiple inheritance)* oznacza, że klasa dziedziczy po 2 lub więcej niezwiązanych ze sobą klasach.
- Dziedziczenie po "babce" i "matce" nie jest jeszcze wielodziedziczeniem.
- Przykład:



- DrawableString dziedziczy po klasach *Point* i *String*.

Wielodziedziczenie — przykład

```
class DrawableString inherits from Point, String {
atributes: /* tylko dziedziczone z nadklas */
methods:   /* tylko dziedziczone z nadklas */
}

DrawableString dstring
dstring.setX(10)
dstring.append("Pójdź, kiń-że tę chmurność...")
```

`dstring` odpowiada na komunikaty z protokołów klas `Point` i `String`.