Modelowanie i Programowanie Obiektowe

POLITECHNIKA
POZNAŃSKA

Wykład III: Programowanie obiektowe na platformie .Net

28 październik 2013

Programowanie Obiektowe

- **Metodologia** tworzenia programów
- Program jest to **zbiór elementów** zwanych obiektami, które łączą stan i zachowanie
- Obiekty **komunikują się** pomiędzy sobą w celu wykonania określonych (zdefiniowanych) zadań

Co daje programowanie obiektowe?

- Programowanie – modelowanie rzeczywistości
- Program to ciąg instrukcji zrozumiałych dla CPU
- Paradygmaty programowania: deklaratywne, zdarzeniowe, proceduralne, obiektowe...
- Ludzkie poznawanie świata:
  - Obiekty i ich interakcja
  - Uogólnianie i klasyfikacja obiektów
- Obiekt wykonuje swoje zadania bez ujawniania szczegółów realizacji operacji innym obiektom

Program obiektowy na jednym slajdzie

- wszystko jest obiektem
- program wykonuje zadanie dzięki interakcji obiektów między sobą
- obiekt posiada swój typ (klasę)
- obiekt ma swoją pamięć
- obiekt może się składać z innych obiektów
- obiekt udostępnia swoje metody (funkcje)
- obiekt w programie <-> obiekt rzeczywisty

Klasa czy obiekt?

- Klasa jest typem danych
- Opisuje zbiór obiektów o takiej samej charakterystyce
- Obiekt – konkretna realizacja (egzemplarz) klasy
- Analogie:
  - klasa – szablon formularza, obiekt – wypełniony formularz
  - klasa – projekt techniczny samochodu, obiekt – konkretny egzemplarz samochodu

Budowa klasy

- Klasa składa się z (ma składowe typu):
  - pól (typów prostych i złożonych)
  - metod
- Pola przechowują stan obiektu
- Pola są obiektami
- Metody służą do interakcji z obiektem i zmieniania jego stanu (pól)
- Metody mogą przyjmować jako parametr inne obiekty lub zwracać inne obiekty

Przykład klasy - 1

```
class PIT-2 {
   NumerNIP;
   Pesel;
   Imię;
   Nazwisko;
   Adres;

   SprawdźPisownię () { ... };
   ZmienAdres (nowyadres) { ... };
   Data PodajDatęUrodzenia () { ... };
}
```

Przykład klasy - 2

```
class BMW-M3 {
   Silnik;
   NumerNadwozia;
   NumerSeryjny;
   RokProdukcji;

   UruchomSilnik () { ... };
   UstawBieg (numerBiegu) { ... };
}
```

Przykład programu – 1

```
public class Osoba {
    Imię;
    Nazwisko;
    PESEL;
}

public class Rejestr {
    List<Osoba> Uczestnicy;

    DodajOsobe (imie, nazwisko, pesel) {
        Uczestnicy.Add (new Osoba (imie, nazwisko, pesel));
    }

    UsunOsobe (osoba) {
        Uczestnicy.Remove (osoba);
    }
}
```

Przykład programu – 2

```
public class Forum {
     List<Watek> watki;

     DodajNowyWatek (tytul, tresc) {
          watki.Add (new Watek (tytul, tresc));
     }
}

public class Watek {
     Post PostPoczatkowy;
     List<Post> Odpowiedzi;

     Watek (tytul, tresc) {
          PostPoczatkowy = new Post (tytul, tresc, null);
     }

     DodajNowaOdpowiedz (tytul, tresc) {
          Odpowiedzi.Add (new Post (tytul, tres, PostPoczatkowy));
     }
```

Przykład programu – 2 cd.

```
public class Post {
     Tytul;
     Tresc;
     PostBazowy;
     List<Post> Odpowiedzi;

     Post (tytul, tresc, post) {
          Tytul = tytul;
          Tresc = tresc;
          PostBazowy = post;
     }

     DodajOdpowiedz (tytul, tresc) {
          Odpowiedzi.Add (new Post (tytul, tresc, this));
     }
}
```

Bezpieczeństwo i prostota

- Obiekty mogą być duże i złożone – nie powinno to komplikować ich implementacji
- Operacje wykonywane przez obiekt (czasami) wymagają wykonania innych jego operacji
- Nie wszystkie operacje/działania obiektu powinny być widoczne dla wszystkich
- Dla wykonania operacji obiekt może potrzebować zapisać pole, którego przypadkowa modyfikacja jest groźna (np. saldo konta)
- Obiekt definiuje kto ma dostęp do jego składowych
- Ukrywanie implementacji – prostszy program

Zakresy widoczności

- składowe klasy mają ograniczony zasięg
- „nie każdy musi (powinien) wiedzieć wszystko”
- metoda działa na polach swojego obiektu
- stan innych obiektów zmieniają ich własne metody
- typy widoczności:
  - private – widoczne tylko dla danego

Zakresy widoczności - przykład

```
public class KontoBankowe {
   private double saldo;

   public Wplata (double kwota) {
      if (kwota <= 0)
         ERROR (zła kwota);
      else
         saldo += kwota;
   }
}
```

Tworzenie / niszczenie obiektów

- Wyróżnia się 2 specjalne metody:
  - konstruktor – uruchamiany przy tworzeniu obiektu,
  - destruktor – uruchamiany przy niszczeniu obiektu,
- Konstruktor
  - często obiekt nie może istnieć (nie ma sensu), o ile nie zostaną ustawione początkowe wartości jego Pól,
  - obiekt może potrzebować stworzyć swoje obiekty składowe, aby mógł poprawnie działać

Tworzenie / niszczenie obiektów cd.

- destruktor
  - często obiekt potrzebuje składowych, które będą istniały całe jego „życie”,
  - w trakcie istnienia obiektu często tworzy on inne obiekty, które są mu potrzebne przez bliżej nieokreślony czas,
  - destruktor jest specjalną metodą, wołaną dla każdego obiektu przy usuwaniu go z pamięci
  - pozwala usunąć składowe obiekty, których nie było możliwości lub potrzeby usunięcia wcześniej

Charakterystyka programów obiektowych

- **Kompozycja** – tworzenie programów oraz obiektów poprzez organizowanie ich z innych programów (podprogramów) oraz obiektów składowych.
- **Kapsułkowanie (hermetyzacja)** – zamknięcie obiektu na swobodne modyfikacje z zewnątrz, ukrywanie szczegółów obiektu w celu jedynie kontrolowanego dostępu do danych obiektu – zakresy widoczności
- **Dziedziczenie** – wykorzystanie już zdefiniowanych klas jako podstaw do tworzenia klas bardziej wyspecjalizowanych
- **Polimorfizm** – wywoływanie metody na podstawie

Kapsułkowanie

- odizolowanie danych i operacji obiektu od innych obiektów
- bardzo złożony obiekt może być bardzo prosty w użyciu (zaledwie kilka publicznych metod),
- ułatwia zarządzanie i utrzymanie kodu,
- ułatwia konstruowanie obiektów złożonych i skomplikowanych systemów (setki obiektów)
- uniemożliwia przypadkowe niechciane zmiany w obiekcie, które mogą prowadzić do błędów (m.in. błędy programistyczne)
- klasa *Telewizor* – bardzo złożone urządzenie, jednak publicznie udostępnia jedynie metody

Dziedziczenie

- wykorzystanie klasy podstawowej (bazowej) do stworzenia nowej, bardziej specjalizowanej klasy:
  - Klasa bazowa: *Osoba*
  - Klasy dziedziczące: *Student*, *Pracownik_PP*, *Admin*
- zachowywane są wszystkie pola i metody klasy bazowej
- traci się dostęp do składowych `private`
- jest specjalny zakres widoczności `protected` – widoczne w klasach dziedziczących, jednak niedostępne z zewnątrz obiektu

Dziedziczenie - przykład

```
class Samolot {
    PrędkośćMax;
    PułapMax;
    ZasięgMax;

    Wystartuj ()
      { ... };
    Ląduj () { ... };
}
```

```
class Myśliwiec : Samolot {
    RakietyAA;
    PociskiAS;
    DziałkoPokładowe;

    OdpalRakietę () { ... };
    UruchomDziałko () { ... };
}
```

Interfejs

- opisuje, jaką funkcjonalność (jakie metody) musi zawierać klasa dziedzicząca po interfejsie
- są sposobem na wymuszenie istnienia metod w klasie, która implementuje (dziedziczy po) interfejs
- nie zawiera implementacji metod ani żadnych pól, zawiera jedynie nagłówki, co umożliwia abstrahując od szczegółów korzystać z klas, które będą po nim dziedziczyć
- pozwalają traktować różne klasy w ten sam sposób

Interfejs - przykład

```
interface IPojazd {
   UruchomSilnik ();
}
```

```
class Samochód : IPojazd {
   UruchomSilnik () {
      // silnik benzynowy
   }
}

class Czołg : IPojazd {
   UruchomSilnik () {
      // silnik diesla
   }
}
```

Polimorfizm

- klasy, które są ze sobą w relacji dziedziczenia, często muszą wykonywać tę samą operację w inny sposób (np. uruchamianie silnika)
- w takiej sytuacji każdą klasę trzeba obsłużyć inaczej, tj. sprawdzić, jakiego jest ona typu i wywołać odpowiednią dla niej wersję metody
- polimorfizm pozwala użyć klasy nadrzędnej jako uchwytu do jednej z klas podrzędnych, jednocześnie zapewniając wołanie metody zaimplementowanej w klasie podrzędnej

Polimorfizm – referencja

- obiekt jest instancją klasy, konkretnym egzemplarzem danego szablonu
- jak odwołać się (znaleźć) obiekt?
- jak pokazać komuś o jaki obiekt nam chodzi?
- referencja – uchwyt, wskaźnik na obiekt, jego adres
- każdy obiekt ma przynajmniej jedną ważną referencję (w przeciwnym razie jest usuwany)
- przekazywanie obiektu do funkcji – kopiowanie referencji (a nie kopiowanie obiektu), zatem zmiany będą widoczne po zakończeniu funkcji

Polimorfizm cd.

- referencją do obiektu klasy pochodnej może być zmienna dowolnej klasy bazowej lub interfejs implementowany przez daną klasę
- wczesne wiązanie (statyczne) – ustalanie typu w czasie kompilacji
- późne wiązanie (dynamiczne) – ustalanie typu obiektu w czasie uruchomienia
- referencja klasy nadrzędnej może przechowywać uchwyt do klasy dziedziczącej
  - drzewo hierarchii dziedziczenia

Polimorfizm - przykład

```
class Pies {
    Głowa;
    Tułów;
    Ogon;

    virtual Szczekaj () {
        Write(„szczekam”);
    }
}
```

```
class Pitbul : Pies {
    override Szczekaj () {
        Write („szczekam
          groźnie”);
    }
}

class York : Pies {
    override Szczekaj () {
        Write („ledwo
          miauczę”);
    }
}
```

Polimorfizm - przykład cd.

```
TestPsów () {
    Pies piesek = new Pies ();
    piesek.Szczekaj ();              --> szczekam

    Pitbul kiler = new Pitbul ();
    kiler.Szczekaj ();                  --> szczekam groźnie

    piesek = kiler;
    piesek.Szczekaj ();              --> szczekam groźnie

    Pitbul kiler2 = piesek;                --> Błąd!!
    kiler2 = (Pitbul) piesek;           --> Poprawne
}
```

Klasa abstrakcyjna

- klasa bazowa, która zawiera implementację pewnych, jednak nie wszystkich metod
- nie można tworzyć obiektów klasy abstrakcyjnej
- ma metody abstrakcyjne – ich implementacja zostanie dostarczona w klasie dziedziczącej
- po zaimplementowaniu wszystkich metod abstrakcyjnych w klasie pochodnej można tworzyć obiekty nowo powstałej klasy

Składowe statyczne

- w definicji klasy poprzedzane przez `static`
- nie należą do żadnego obiektu, należą bezpośrednio do klasy
- istnieje tylko jeden ich egzemplarz – nie są tworzone nowe wraz z tworzeniem obiektów

```
class Samochód {
   static LiczbaWyprodukowanych;
}

class Fabryka {
ProdukujSamochód () {
   Samochód fiat = new Samochód ();
   Samochód.LiczbaWyprodukowanych += 1;
}
}
```

Składowe stałe

- statyczne i niezmienne w ciągu całego działania programu

```
class Matematyka {
  const Pi = 3.1415;
}
```