Programowanie obiektowe

Wykład 4.

Marcin Młotkowski

15 marca 2024

Plan wykładu

1. Właściwości
2. Interfejsy
3. Model obiektowy
   - Klasa podstawowa
   - Jak odróżnić obiekty
   - Własne kolekcje
   - Obiekty i wartości

Plan wykładu

Z czego składa się obiekt

- pola;
- metody;
- właściwości.

Motywacje

- pola wyliczane (tylko do odczytu): `wiek` czy `godzina`;
- kontrola przypisania: `miesiąc`

Niedogodności stosowania pól

- jeśli pole jest publiczne wszyscy mogą czytać i modyfikować pola;
- nie można kontrolować podstawianych wartości;
- nie można ustalać wartości "w locie" w trakcie odwołania do pola.

Częściowe rozwiązanie

Akcesory

Dla każdej zmiennej o nazwie Variable tworzymy dwie metody

- SetVariable(val)
- GetVariable()

Właściwości (propercje)

- trochę przypomiają zmienne a trochę metody;
- "z zewnątrz" przypominają pola;
- "od wewnątrz" przypominają metody.

Przykład implementacji właściwości

```
class Data
{
    int miesiac, dzien;
    public int Miesiac {
        get { return miesiac; }
        set {
            if (0 < value && value < 13)
                miesiac = value;
            else miesiac = 1;
            }
    }
    public int Dzien {
        get {
            return dzien;
        }
    }
}
```

Zastosowanie

```
Data d = new Data();
d.Miesiac = 3;
Console.WriteLine(d.Dzien);
```

Zastosowanie

```
Data d = new Data();
d.Miesiac = 3;
Console.WriteLine(d.Dzien);
d.Dzien = 23;
```

Singleton po raz kolejny

```
sealed class Singleton
{
    Singleton() {}
    static Singleton instance;
    public static Singleton Instance()
    {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
}
```

Modyfikacja: właściwość

```
public static Singleton Instance
{
    get
    {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
}
```

Plan wykładu

Motywacje

Co można robić z obiektami

- zapisywać/odczytywać do/z pliku;
- porównywać (w kolekcjach uporządkowanych);
- klonować;
- i wiele innych rzeczy ...

Przykład

```
object[] table = new object[2];
table[0] = new Punkt();
table[1] = new Tramwaj();
```

Przykład

```
object[] table = new object[2];
table[0] = new Punkt();
table[1] = new Tramwaj();
```

Jak wymusić, aby klasy implementowały metodę Clone()?

Jaki ma typ tablica table?

Interfejs

- Interfejs definiuje typ, nie klasę.
- Interfejs deklaruje pola i metody (bez implementacji)
- Klasy mogą implementować wiele interfejsów

Typ to nie klasa!

Implementacja listy, wersja A

```
class ListaA
{
    private object[] lista;
    public void Append(object elem) { .... }
}
```

Implementacja listy, wersja B

```
class ListaB
{
    class Elem
    {
        public object val;
        public Elem next;
    }
    public void Append(object elem) { ... }
}
```

Wspólny „wygląd” klas

```
interface Lista
{
    public void Append(object);
}
```

Wspólny „wygląd” klas

```
interface Lista
{
    public void Append(object);
}
class ListaA: Lista
{ ... }

class ListaB: Lista
{ ... }
```

Inny przykład

```
interface ICloneable
{
    object Clone();
}
```

Implementacja interfejsu

```
class Samochod : Pojazd, System.ICloneable
{
    ...
    public object Clone()
    {
        return this.MemberwiseClone();
    }
    ...
}
```

Implementacja interfejsu

```
class Samochod : Pojazd, System.ICloneable
{
    ...
    public object Clone()
    {
        return this.MemberwiseClone();
    }
    ...
}
```

Przykłady użycia

```
Cloneable[] tablica = new ICloneable[10];
tablica[0] = new Samochod();
```

Nie można tworzyć obiektów

~~tablica[1] = new ICloneable();~~

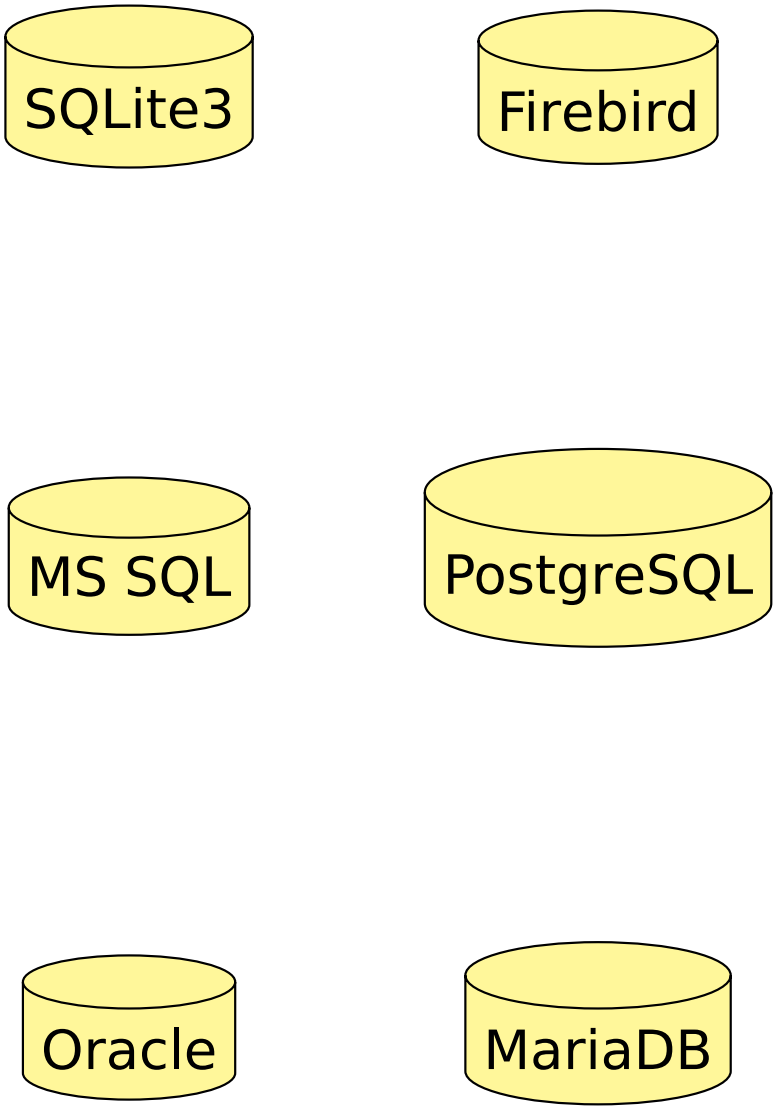
Porównywanie obiektów

```
interface IComparable
{
    public int CompareTo(object obj);
}
```

Kolekcja uporządkowana

```
class OrderedCollection
{
    void Add(IComparable elem)
    {
        ...
    }
}
```

Bazy danych



Implementacja sterowników

- Twórcy silników baz danych mogą dostarczać własnych sterowników dostępu do baz danych;
- sterowniki powinne implementować co najmniej pola i metody zdefiniowane w interfejsie `System.Data.IDbConnection` i pochodnych.

Interfejs

```
namespace System.Data;

interface IDbConnection
{
    void Close();
    void Open();
    IDbCommand CreateCommand();
    ...
}
```

Fabryka obiektów

```
class WykladPrzyklad {
  enum RDBMS = { mysql, sqlite, oracle }
  public static IDBConnection SQLDriver(RDBMS typ, string cs)
  {
    IDbConnection conn;
    switch (typ) {
      case mysql:
        conn = new MySql.Data.MySqlClient.MySqlConnection(cs);
        break ;
      case sqlite:
        conn = new Mono.Data.SqliteClient.SqliteConnection(cs)
        break ;
      case oracle:
        conn = new System.Data.OracleConnection(cs);
        break ;
      }
    return conn;
  }
}
```

Zastosowanie Fabryki

```
RDBMS wybor;

IDbConnection connection =
        WykladPrzyklad.Connection(wybor);

connection.Open();
IDbCommand cmd = connection.CreateCommand();
cmd.CommandText = "SELECT * FROM Studenci";
IDataReader rd = cmd.ExecuteReader();
```

Interfejsy generyczne

```
interface IComparable<T>
{
    int CompareTo(T obj);
}
```

Przykład

```
class Informacja {
    public System.DateTime create;
    public String Nazwisko;
}
```

Lista uporządkowana

```
class Lista<T>  {
    Lista<T> next;
    protected T val;
    public void Add(T val) {
        if (this.val < val) ...
        else { ... }
}
```

Lista uporządkowana

```
class Lista<T> where T : IComparable<T> {
    Lista<T> next;
    protected T val;
    public void Add(T val) {
        if (this.val.CompareTo(val) > 0) ...
        else { ... }
}
```

Przykład

```
class Informacja {
    public System.DateTime create;
    public String Nazwisko;

}
```

Nie umiemy porównac obiektów Informacja

```
Lista<Informacja> lista = new Lista<Informacja>();
lista.Add(new Informacja());
```

Przykład

```
class Informacja : IComparable<Informacja>{
    public System.DateTime create;
    public String Nazwisko;

}
```

Przykład

```
class Informacja : IComparable<Informacja>{
    public System.DateTime create;
    public String Nazwisko;

    public int CompareTo(Informacja i) {
        if (i.create != this.create)
            return i.create.CompareTo(this.create);
        return i.Nazwisko.CompareTo(this.Nazwisko);
}
```

Plan wykładu

Model obiektowy

Zbiór zwyczajów i zaleceń dotyczących programowania w danym języku.

Klasa System.Object

Klasa `Object`[a] jest nadklasą wszystkich innych klas, nawet jeśli jawnie nie jest to zadeklarowane.

[a] zamiennie można pisać `object`

Metody klasy Object

```
class Object
{
    public virtual bool Equals(Object obj);
    public virtual int GetHashCode();
    public virtual string ToString();
```

Metody klasy Object

```
class Object
{
    public virtual bool Equals(Object obj);
    public virtual int GetHashCode();
    public virtual string ToString();
    public Type GetType();
    ...
}
```

Przypomnienie

```
class Samochód : Pojazd
{

}

Console.WriteLine(new Samochód());
```

Przypomnienie

```
class Samochód : Pojazd
{

}

Console.WriteLine(new Samochód());
```

Otrzymamy

'Samochód'

Chcemy

'auto marki: Syrena'

Implementacja

```
class Samochod : Pojazd
{
    public override string ToString()
    {
        return String.Format("auto marki: {0}",
                             this.marka);
    }
}
```

Zastosowanie

```
bryka = new Samochod();
// zamiast: Console.WriteLine(bryka.info());
Console.WriteLine(bryka);
Console.WriteLine("To jest " + bryka);
```

```
class Informacja {
    string n; int i;
    public Informacja(string n, int i)
    {
        this.i = i;
        this.n = n;
    }
}
```

```
class Informacja {
    string n; int i;
    public Informacja(string n, int i)
    {
        this.i = i;
        this.n = n;
    }
}
Informacja i1 = new Informacja("abc", 3);
Informacja i2 = new Informacja("abc", 3);
```

```
class Informacja {
    string n; int i;
    public Informacja(string n, int i)
    {
        this.i = i;
        this.n = n;
    }
}
Informacja i1 = new Informacja("abc", 3);
Informacja i2 = new Informacja("abc", 3);
```

Czy i1 == i2? fałsz

Czy i1.Equals(i2)? fałsz

Własna metoda Equals

```
class Informacja {
    public string n; int i
    public Informacja(string n, int i)
    {
        this.i = i;
        this.n = n;
    }
    public override bool Equals(Object o)
    {
        var inf = o as Informacja;
        return this.n == inf.n;
    }
}
```

Równość obiektów

```
Informacja i1 = new Informacja("abc", 3);
Informacja i2 = new Informacja("abc", 3);
```

Czy

```
  i1.Equals(i2)
```

prawda

Równość obiektów

```
Informacja i1 = new Informacja("abc", 3);
Informacja i2 = new Informacja("abc", 3);
```

Czy

```
  i1.Equals(i2)
```

prawda

Czy

```
  i1 == i2
```

fałsz

Równość

==

Operator == korzysta z metody GetHashCode().

```
public override int GetHashCode() {
    return this.n.GetHashCode();
}
```

Badanie klasy obiektu

```
class Samochód { ... }
class Informacja { ... }

Object o;

if (o.GetType() == typeof(Informacja))
    ...
```

Własne kolekcje

```
class Lista<T>
{
    ...
}
Lista list;
for(int i = 0; i < list.Length; i++)
    System.Console.WriteLine(list[i]);
```

Implementacja dostępu indeksowanego

```
class Lista<T>
{
    Lista<T> next;
    protected T val;
    public T this[int indeks] {
      get {
        if (indeks == 0) return val;
        return this.next[indeks - 1];
      }
    }
}
```

Przykłady użycia

```
Lista<int> list = new Lista<int>();
list.Add(4);
list.Add(8);
System.Console.WriteLine(list[2]);
for (int i = 0; i < list.Length; i++)
    System.Console.WriteLine(list[i]);
```

Prawdziwe kolekcje

```
foreach(int e in list)
    System.Console.WriteLine(e);
```

Implementacja

- Zaprogramujemy klasę `Lista<T>` implementującą interfejs `System.Collections.IEnumerable`;
- interfejs `IEnumerable` wymaga implementacji metody `IEnumerator GetEnumerator();`
- zaprogramujemy enumerator `ListEnum<T>` implementujący interfejs `IEnumerator`;
- zaprogramujemy klasę `Element<T>`.

Implementacja elementów listy

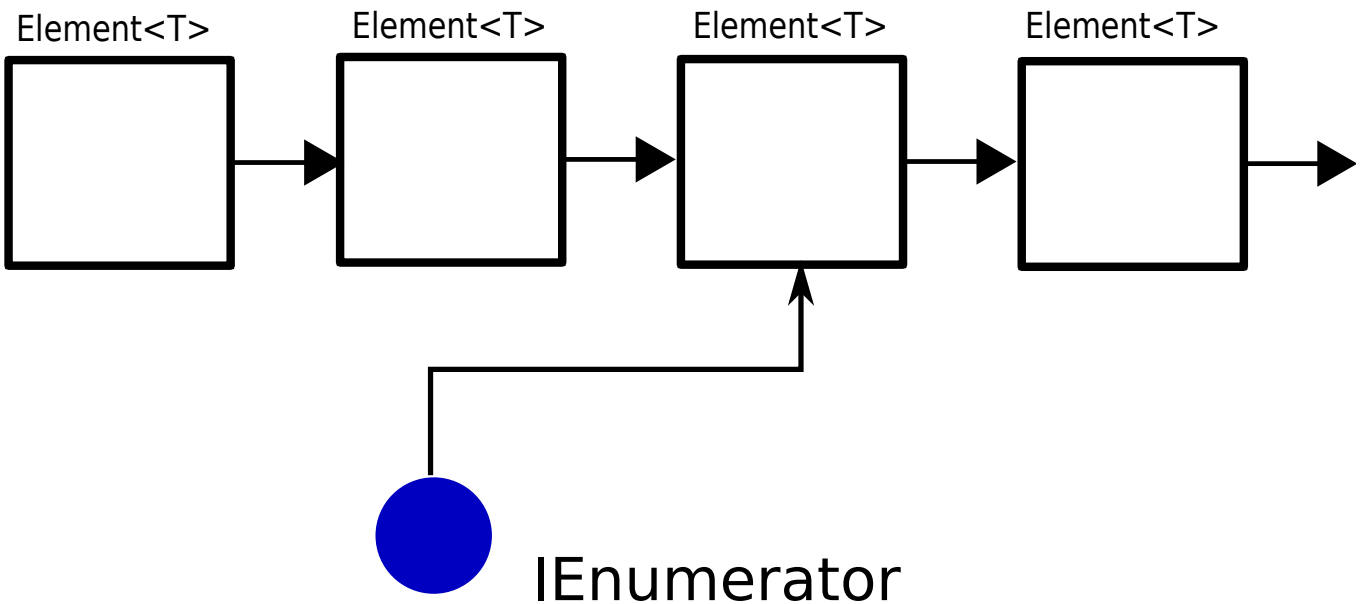
```
class Element<T>
{
    public T val;
    public Element<T> next;
}
```

Implementacja listy

```
using System.Collections;

class Lista<T> : IEnumerable
{
    Element<T> lista;
    public void Add(T val) { ... }
    // Implementacja interfejsu
    public IEnumerator GetEnumerator()
    {
        return new ListEnum<T>(lista);
    }
}
```

Schemat


Element<T>
Element<T>
Element<T>
Element<T>
IEnumerator
MoveNext()
Current { get; }

Algorytm działania enumeratora

Schemat

- zainicjowanie przeglądania kolekcji;
- zwracanie kolejnych elementów;
- sygnał końca kolekcji.

Implementacja enumeratora

```
class ListEnum<T> : IEnumerator
{
    Element<T> lista;
    public ListEnum(Element<T> lista) { ... }
    public bool MoveNext() { ... }
    public object Current { ... }
    public void Reset() { ... }
}
```

Implementacja enumeratora

```
class ListEnum<T> : IEnumerator
{
    Element<T> lista;
    public ListEnum(Element<T> lista)
    {
        this.lista = lista;
    }
    public bool MoveNext() { ... }
    public object Current { ... }
    public void Reset() { ... }
}
```

Implementacja enumeratora

```
class ListEnum<T> : IEnumerator
{
    Element<T> lista;
    public ListEnum(Element<T> lista) { ... }
    public bool MoveNext()
    {
        if (this.current == null) this.current = this.lista;
        else this.current = this.current.next;
        return this.current != null;
    }
    public object Current { ... }
    public void Reset() { ... }
}
```

Implementacja enumeratora

```
class ListEnum<T> : IEnumerator
{
    Element<T> lista;
    public ListEnum(Element<T> lista) { ... }
    public bool MoveNext() { ... }
    public object Current {
        get {
            return current.val;
        }
    }
    public void Reset() { ... }
}
```

Implementacja enumeratora

```
class ListEnum<T> : IEnumerator
{
    Element<T> lista;
    public ListEnum(Element<T> lista) { ... }
    public bool MoveNext() { ... }
    public object Current { ... }
    public void Reset()
    {
        this.current = this.lista;
    }
}
```

Zastosowanie

```
Lista<int> list = new Lista<int>();
list.Add(4);
list.Add(8);
foreach(int e in list)
    Console.WriteLine(e);
```

Gdzie przechowujemy dane

- obiekty
- wartości

Wartości (value types)

- Typy podstawowe: `int`, `float`, `bool`, etc.
- zmienne przechowują wartości;
- przypisanie zmiennej innej zmiennej oznacza skopiowanie wartości;
- wartości są przechowywane na stosie;
- dla typów podstawowych istnieją odpowiednie klasy (typy referencyjne), np.

bool	System.Boolean
int	System.Int32
float	System.Single

Obiekty (refrencje)

- zmienne przechowują referencje do obiektów;
- przypisanie zmiennej zmiennej oznacza skopiowanie referencji, nie wartości;
- obiekty przechowywane są na stercie.

Jawna konwersja typów

Zamiana wartości na obiekt (boxing):

```
int i = 123;
object o = (object)i;
```

Zamiana obiektu na wartość (unboxing):

```
int i = 123;
object o = (object)i;
int j = (int)o;
```

Niejawna konwersja

Niekiedy można pomijać operator konwersji.

Zamiana wartości na obiekt (boxing):

```
int i = 123;
object o = i;
```

Zamiana obiektu na wartość (unboxing) prawie zawsze wymaga jawnej konwersji:

```
int i = 123;
Object o = i;
int j = (int)o;
```

Bezpieczna konwersja

```
Object o;

if (o is Pojazd)
    Pojazd p = (Pojazd)o;
```

Albo:

```
Object o;

if (o is Pojazd)
    Pojazd p = o as Pojazd;
```