Politechnika Wrocławska

Platformy programistyczne

Charakterystyka obiektowości, wyrażenia lambda, język LINQ

Dr inż. Radosław Idzikowski

Katedra Automatyki, Mechatroniki i Systemów Sterowania
Wydział Informatyki i Teleinformatyki

15 marca 2024

HR EXCELLENCE IN RESEARCH

Dziedziczenie

- dziedziczenie jest pojedyncze (tzn. może odbywać się tylko po jednej klasie bazowej),
- dziedziczenie dostępne jest wyłącznie dla klas,
- klasa lub struktura może implementować wiele interfersjów,
- modyfikatory dostępności:
  - `public` – dostępne dla wszystkich,
  - `protected` – dostępne tylko wewnątrz typu lub dla jego klas potomnych,
  - `internal` – dostępne tylko wewnątrz `assembly` (lub zaprzyjaźnionego),
  - `private` – dostępne tylko z wewnątrz typu (**domyślny modyfikator**).

```
class Animal
{
    public string Name { get; set; }
}
class Dog : Animal
{
    public int Age { get; set; }
}
internal class Program
{
    static void Main(string[] args)
    {
        Dog dog = new Dog() {Name = "Rex",Age = 2 };
        Console.WriteLine(dog.Name);
        Console.WriteLine(dog.Age);
        Console.ReadKey();
    }
}
```

Dziedziczenie w praktyce

lokalizacja	`public`	`protected internal`	`proteckted`	`internal`	`private protected`	`private`
w ramach klasy	✓	✓	✓	✓	✓	✓
klasa pochodna (`assembly`)	✓	✓	✓	✓	✓	×
klasa nie pochodna (`assembly`)	✓	✓	×	✓	×	×
klasa pochodna	✓	✓	✓	×	×	×
klasa nie pochodna	✓	×	×	×	×	×

Polimorfizm

Zmienna typu bazowego może się odnosić do obiektu będącego jego podklasą.

```
class Animal
{
    public string name;
    public Animal(string _name)
    { name = _name; }
}
class Dog : Animal
{
    public int age;
    public Dog(string _name, int _age) : base(_name)
    { age = _age; }
}
public static string GetName(Animal animal)
    { return animal.name; }
```

Rzutowanie i konwertowanie referencji

- niejawne rzutowanie w górę:

```
Animal anim = new Dog() { Name = "Rex", Age = 2 };
```

- jawne rzutowanie w dół:

```
Dog dogy = (Dog)a;
```

- operator as – rzutowanie w dół, ale w przypadku niepowiedzenia zwraca `null`.
- operator `is` – sprawdzenie czy konwersja referencyjna przejdzie pomyślnie:

```
if (a is Dog){
    Dog dogy = (Dog)a;
    Console.WriteLine(dogy.Age);}
```

- zmienna wzorcowa:

```
if (a is Dog dogy)
    Console.WriteLine(dogy.Age);
```

Wirtualne składowe funkcyjne

- metody,
- własności,
- indeksatory,
- zdarzenia.

Typ object – przypomnienie

```
public class Stack
{
    int x;
    object[] objects;
    public Stack(int size = 10)
    { x = 0; objects = new object[size]; }
    public void Push (object obj) { objects[x++] = obj;}
    public object Pop () { return objects[--x]; }
}
static void Main()
{
    Stack stack = new Stack();
    stack.Push(1.5f);
    float number = (float)stack.Pop();
    Console.WriteLine(number);
}
```

Typy generyczne

```
public class Stack<T>
{
    int x;
    T[] data;
    public Stack(int size = 10)
    { x = 0; data = new T[size]; }
    public void Push(T obj) { data[x++] = obj; }
    public T Pop() { return data[--x]; }
}
static void Main()
{
    var stack = new Stack<float>();
    stack.Push(1.5f);
    float number = stack.Pop();
    Console.WriteLine(number);
}
```

Delegaty

- delegat – obiekt, który wie jak wywołać metodę,
- koncepcyjnie podobne do wskaźników na funkcję w C++, ale dbają o bezpieczeństwo typów,
- pozwalają, aby metody były przekazywane jako parametry,
- często używane do definiowania metody typu *callback*,
- delegaty mogą być łączone tak, aby wiele metod było wywołanych przy jednym zdarzeniu.

```
delegate int Transformer(int x);
```

Prosty delegat

```
delegate int Transformer(int x);
static int Square(int x) { return x * x; }
static void Main()
{
    Transformer t = Square;
    Console.WriteLine(t(4));
    Console.ReadKey();
}
```

plug-in methods

```
delegate int Transformer(int x);
static int Square(int x) { return x * x; }
static int Cube(int x) { return x * x * x; }
static void Transform(int[] values, Transformer t)
{
    for (int i = 0; i < values.Length; i++)
        values[i] = t(values[i]);
}
static void Main()
{
    int [] arr = { 1, 3, 4, 5 };
    Transformer t = Square;
    Transform(arr, t);
    foreach (int x in arr) Console.Write(x + "␣");
    Transform(arr, Cube);
}
```

Wyrażenie lambda

- niezwana metoda przypisana w miejsce delegatu,
- w praktyce jest konwertowana na etapie kompilacji do instancji delegatu lub drzewa wyrażeń.

```
(parametry)  => wyrazenie lub blok kodu

x    =>          x * x;

Transformer sqr= x    => {return x * x;}
```

Politechnika Wrocławska

LINQ

- `LINQ` – *Language Integrated Query*,
- zestaw własności języka i frameworka, który pozwala na tworzenie ustrukturalizowanych zapytań do kolekcji obiektów,
- dodane już w `C# 4.0`
- podstawowe pojęcia:
  - sekwencja — dowolny obiekt implementujący `IEnumerable<T>`
  - elementy kolekcji,
  - operator zapytania (query operator) – metoda, która przekształca sekwencję.
- istnieje gotowy zestaw standardowych operatorów – wszystkie zaimplementowane są jako metody rozszerzające.

Wyrażenia zapytań

```
var new_names = from n in names
                where n.Length > 3
                orderby n
                select n.ToUpper();
```