Wykład 2
Klasy

dr inż. Maciej Kusy

Katedra Podstaw Elektroniki
Wydział Elektrotechniki i Informatyki
Politechnika Rzeszowska

Programowanie w języku C#

Plan wykładu

- Klasy, obiekty, składowe klasy, przykładowa definicja klasy
- Modyfikatory dostępu
- Konstruktor, inicjalizacja bezpośrednia za pomocą parametrów, wywołanie konstruktora
- Cechy konstruktora
- Typy anonimowe
- Słowo kluczowe (referencja) **`this`**
- Przekazywanie argumentów do metody
- Parametry z modyfikatorem **`out`**
- Cechy programowania obiektowego
- Hermetyzacja za pomocą właściwości
- Używanie składowych statycznych, klasy statyczne
- Struktury i jej podstawowe właściwości

Klasy i obiekty

Klasa – najbardziej podstawowy typ języka C#;
– struktura danych łącząca stan (pola) z działaniami (metody, zdarzenia) w jedną całość;
– stanowi definicję tworzonych dynamicznie instancji zwanych obiektami;
– umożliwia korzystanie z dziedziczenia i polimorfizmu.

Obiekt – egzemplarz (instancja) danej klasy.

Różnica – przykład:
- klasa, np. **`Pies`** opisuje jakie są psy (imię, rasa, wiek…),
- obiekt klasy to konkretny pies np. **`Reksio`** (imię) jest kundlem (rasa), ma pięć lat (wiek)…

Składowe klasy

Pola – zmienne związane z klasą (statyczne) lub instancją klasy (każdy obiekt zawiera osobną kopię pól klasy: **`this`**).

Metody – składowe implementujące obliczenie lub działanie, które może być wykonywane przez obiekt lub klasę. Metody posiadają: modyfikator dostępu, typ zwracany, nazwę, listę argumentów ujętych w nawiasy okrągłe oraz ciało umieszczone w nawiasach klamrowych.

Właściwości – rozszerzenia pól, mogą być używane dla obiektów jak i klas. Posiadają swoje akcesory, które umożliwiają bezpośredni dostęp do wartości pól.

Zdarzenia – składowe umożliwiające klasie lub obiektowi zgłaszanie powiadomień.

Definicja klasy – przykład

```
class Pies
{
  private string imie = "Reksio";
  protected string rasa = "kundel";
  public int wiek = 5;

  public void Wypisz()
  {
    System.Console.WriteLine(

    "Pies: {0}, {1}, {2}", imie, rasa, wiek);
  }
}
```

Modyfikatory dostępu:

Słowa kluczowe języka C# – określają, które składowe (pola, metody, właściwości, indeksery, zdarzenia) danej klasy są widoczne przez co mogą być używane przez inne klasy i obiekty.

- **`private`** – składowe są prywatne,
- **`protected`** – składowe są chronione,
- **`public`** – składowe są publiczne,
- dodatkowe: **`internal`**, **`protected internal`**.

Modyfikatory dla klas:

- **`abstract`** – dana klasa stosowana wyłącznie jako klasa bazowa; nie można tworzyć obiektów tej klasy; każda klasa pochodna musi implementować wszystkie metody abstrakcyjne i akcesory.
- **`sealed`** – dana klasa nie może być przedmiotem dziedziczenia.
- **`static`** – dana klasa zawiera wyłącznie składowe statyczne.

Modyfikatory pól:

- **`static`** – pole składowe jest częścią klasy (wspólne dla wszystkich obiektów), a nie któregokolwiek obiektu; wywołanie: **`NazwaKlasy.pole`**
- **`readonly`** – wartość pola można jednorazowo przypisać w deklaracji (jak **`const`**) lub konstruktorze klasy – przekształcenie pola w stałą; późniejsza modyfikacja spowoduje błąd kompilacji.
- **`const`** – wartość pola musi zostać ustawiona w momencie jego definicji; późniejsza modyfikacja spowoduje błąd kompilacji.

Modyfikatory metod:

- **`static`** – metoda jest częścią klasy (wspólna dla wszystkich obiektów), a nie któregokolwiek obiektu; wywołanie: **`NazwaKlasy.Metoda(parametry).`**
- **`virtual`** – metoda wirtualna umożliwiająca zastosowanie polimorfizmu wraz z jej przesłoniętym odpowiednikiem.
- **`override`** – metoda klasy pochodnej przesłania tak samo nazwaną metodę odziedziczoną z klasy podstawowej.
- **`new`** – metoda klasy pochodnej nie przesłania swojego odpowiednika w klasie podstawowej, tylko go nadpisuje.
- **`abstract`** – metoda nie jest zdefiniowana w danej klasie i musi zostać zaimplementowana we wszystkich klasach pochodnych od tej klasy.
- **`sealed`** – metoda ostateczna uniemożliwia dalsze przesłonięcie metody w klasie pochodnej (wymagane **`override`**).
- **`extern`** – implementacja na zewnątrz w innym języku.

Konstruktor – definicja

Konstruktor – metoda składowa służąca do zainicjalizowania obiektu danej klasy i nadania mu wartości początkowych.

```
class Pies
{
    private readonly string imie;
    public string rasa;
    private int wiek;
    public Pies(string i, string r, int w)
    {
        imie = i;
        rasa = r;
        wiek = w;
    }
    public Pies() : this("Reksio", "kundel", 1) {}
}
```

Wywołanie konstruktora – sposób 1

Konstruktor wywoływany jest przy tworzeniu obiektu za pomocą operatora **new**:

```
class Program
{
    static void Main(string [] args)
    {
        Pies p1 = new Pies("Szarik", "wilczur",2);
        Pies p2 = new Pies();
        System.Console.Writeln(p1.rasa);
        System.Console.Writeln(p2.rasa);
    }
}
```

Wywołanie konstruktora – sposób 2

W wersji języka **C# 3.0** została wprowadzona nowa konstrukcja, tzw. inicjalizatory obiektów (*object initializers*). Jeżeli klasa posiada **konstruktor domyślny**, to nadanie wartości **publicznym polom i właściwościom** klasy możliwe jest w miejscu tworzenia obiektu:

```
Pies p1 = new Pies {
                       imie = "Pluto",
                       rasa = "pluszak",
                       wiek = 1
                   };
Pies p2 = new Pies(){
                        imie = "Clifford",
                        rasa = "cartoon",
                        wiek = 3
                    };
```

Cechy konstruktora:

- Nazwa konstruktora jest taka sama jak nazwa klasy.
- Nie zwraca żadnej wartości.
- Może być przeładowywany (przeciążany).
- Umożliwia wywołanie określonego kodu przed przystąpieniem do wykonywania kodu zdefiniowanego w ciele konstruktora na podstawie dwóch inicjalizatorów:
  - **`this(`**`lista arrgumentów`**`)`**
  - **`base(`**`lista arrgumentów`**`)`**
- Jeżeli nie zostanie zdefiniowany, kompilator utworzy konstruktor domyślny bezparametrowy
  Uwaga: pola składowe będące typami prostymi (i **`enum`**) inicjalizowane są wartościami **`0`**, składowe **`bool`** przyjmują **`false`**, składowe **`char`** mają przypisany znak **`\0`**, łańcuchy znaków oraz obiekty referencyjne – wartość pustą **`null`**.
- Może być prywatny (klasa **`System.Math`**).

Typy anonimowe

W języku **C# 3.0** możliwe jest tworzenie obiektów bez definiowania ich typu:

```
var czyToPies = new {
                      imie = "brak",
                      rasa = "nieznana",
                      wiek = 0
                    };
```

W efekcie powstaje typ o trzech publicznych polach składowych tylko do odczytu. Składowe obiektu takiego typu rozpoznawane są przez mechanizm *IntelliSense*.

Nowy sposób tworzenia obiektu wygodny jest przy zwracaniu danych w zapytaniach **LINQ**.

Słowo kluczowe **this**

- Słowo kluczowe określające aktualny egzemplarz obiektu.
- Referencja (wskaźnik) **this** to ukryty parametr przekazywany do każdej niestatycznej metody klasy.
- Każda metoda (funkcja składowa) danej klasy może korzystać z innych metod bądź pól tej samej klasy poprzez referencję **this**:

```
public void NazwijPsa(string imie)
{
    this.imie = imie;
}
```

Słowo kluczowe **this** – zastosowanie

- Przekazanie aktualnego obiektu do innej metody jako parametru:

```
class Pies
{
    public bool Porownaj(Object o)
    {
        return o.Equals(this);
    }
}
```

- Wywołanie innego konstruktora danej klasy na liście inicjalizacyjnej.
- Zastosowanie w mechanizmach indeksowania.

Przekazywanie argumentów do metody

- Typy podstawowe – wartościowe (**`int`**, **`long`**, **`char`**) przekazywane są do metody przez wartość.
  Wartości argumentów-zmiennych nie są modyfikowane.
  Uwaga: wszystkie zmienne muszą mieć przypisaną wartość zanim zostaną użyte (np. przesłane do metody przez wartość)!
- Obiekty typów referencyjnych (np. obiekt typu **`Pies`**) przekazywane są do metody przez referencję.
  Uwaga: argumenty-obiekty są modyfikowane!
- Typy skalarne można przesyłać przez referencję, ale argument metody należy wtedy poprzedzić słowem kluczowym **`ref`**:
  `Metoda(ref typ nazwa);`
  Uwaga: słowo kluczowe **`ref`** musi zostać umieszczone w definicji metody oraz w miejscu wywołania.

Modyfikator **out**

- Zmienne wszystkich typów skalarnych muszą mieć przypisaną wartość zanim zostaną przesłane do metody przez wartość lub referencję.
- Jeżeli konieczne jest ominięcie przypisania w miejscu deklaracji zmiennej, aby wysłać zmienną jako argument do metody należy użyć modyfikatora **out** zarówno w definicji jak i w miejscu wywołania.
- Uwaga: jakakolwiek zmiana tak przesłanego argumentu do metody, spowoduje jego modyfikację na stałe (praca na oryginale).
- Przykład: metoda **TryParse(string s, out z)**, gdzie **z** jest zmienną typu, do którego konwertowany jest łańcuch **s**.

Cechy programowania obiektowego:

- Hermetyzacja
- Dziedziczenie
- Polimorfizm
- Abstrakcja

Żart informatyczny:
Pytanie: Ilu programistów obiektowych potrzeba do wymiany żarówki?

Odpowiedź: Ani jednego, wystarczy powiedzieć żarówce, aby sama się wymieniła.

Hermetyzacja

Hermetyzacja – ukrycie wewnętrznego stanu klasy: tj.: ustawienie pól, metod, właściwości jako prywatne (chronione).

Cel – ochrona przed bezpośrednią modyfikacją wartości wybranych pól.

Rozwiązanie – udostępnianie wartości pól za pomocą metod (**C++**, **Java**) lub właściwości klasy (**Visual Basic**, **C#**), które zwracają wartości pól.

Hermetyzacja za pomocą właściwości

Właściwości – kontrolowanie dostępu (zapis i odczyt) do wartości wewnątrz klasy.

Cechy właściwości:

- dają możliwość zarządzania wewnętrznym stanem klasy,
- umożliwiają bezpośredni dostęp do wartości pól,
- sposób dostępu do stanu klasy przypomina dostęp do pól,
- umożliwiają kompilatorowi generowanie bardziej efektywnego kodu.

Właściwość: akcesory **get** i **set**

```
public class Pies
{
    private string imie;

    public string Imie
    {
        get
        {
            return imie;
        }
        set
        {
            imie = value;
        }
    }
}
```

Bezpośredni dostęp do pola bez naruszenia hermetyzacji

Poprawność przypisywanej wartości

Wywołanie właściwości

Właściwość może być wywoływana na rzecz obiektu klasy (składnia identyczna jak przy odwołaniu się do pola):

```
class Program
{
    static void Main(string [] args)
    {
        Pies p1 = new Pies("Szarik", "wilczur",2);
        Pies p2 = new Pies();
        p1.Imie = "Reksio";
        p2.Imie = "Pluto";
        Console.WriteLn("Psy nazywają się: {0} i {1}",
                        p1.Imie, p2.Imie);
    }
}
```

Akcesor **set**

Akcesor **get**

Używanie składowych statycznych

Zwykłe składowe klasy (pola, metody, właściwości, zdarzenia) mogą być składowymi egzemplarza danej klasy. Są wtedy powiązane z danym obiektem (**`this`**).

Składowe statyczne są elementami klasy. Są użyteczne gdy chcemy posiadać informację wspólną dla wszystkich obiektów klasy (np. oprocentowanie konta bankowego).

Ciekawostka: język C# jest językiem czysto obiektowym. Nie można w nim tworzyć metod globalnych. Efekt ten można uzyskać definiując w klasie metody statyczne.

Uwaga: można zdefiniować konstruktor statyczny. Służy on wyłącznie do inicjalizowania statycznych pól składowych.

Składowe statyczne – uwagi:

- Jeżeli w metodzie nie ma odwołania do żadnego pola składowego, metoda powinna być statyczna (lepsza wydajność).
- Do metod statycznych nie jest przekazywany parametr **`this`**.
- Pola **`const`** są niejawnie polami statycznymi.
- Konstruktor statyczny wywoływany jest niejawnie przed utworzeniem obiektu i przed odwołaniem się do składowych statycznych.
- Konstruktora statycznego nie można wywołać jawnie.
- Konstruktora statycznego nie można przeciążać (jest on bezparametrowy).
- Konstruktor statyczny nie ma modyfikatora dostępu.
- Składowymi statycznymi mogą być również właściwości.

Klasy statyczne

- Jeżeli klasa zdefiniowana jest jako statyczna, wszystkie jej pola, metody, właściwości muszą również być statyczne.
- Klas statycznych nie można dziedziczyć.
- Nie można tworzyć obiektów klas statycznych.
- Klasa statyczna może posiadać wyłącznie jeden konstruktor i musi być on statyczny.
- Zdefiniowanie klasy jako statycznej zapobiega przypadkowemu stworzeniu obiektu.
- **Wskazówka**: jeżeli wszystkie składowe klasy są statyczne, to klasę przechowującą powinno się również zadeklarować jako statyczną. Obrazuje ona wtedy specyficzną funkcjonalność.

Struktury

Struktura, podobnie jak klasa, jest typem zdefiniowanym przez użytkownika. Deklaruje się ją przy użyciu słowa kluczowego **`struct`**.

```
public struct Punkt
{
    private double wspX, wspY;
    public Punkt(double x, double y)
    {
        wspX = x;
        wspY = y;
    }
    public void Wypisz()
    {
        Console.WriteLine("{0}, {1}",wspX, wspY);
    }
}
```

Podstawowe właściwości struktur

- Mogą zawierać pola, właściwości, metody (również konstruktory), operatory, zdarzenia, indeksery.
- Są typami wartościowymi, które niejawnie dziedziczą po wbudowanej klasie bazowej **`System.ValueType`**.
- Nie mogą dziedziczyć po klasach ani być typem dziedziczonym ale mogą implementować interfejsy.
- Ich konstruktory muszą posiadać przynajmniej jeden argument.
- Wszystkie pola składowe muszą być zainicjalizowane w konstruktorze struktury.
- Obiekty struktur składowane są na stosie.
- Powinno się ich używać w przypadku małych i prostych typów.