

# Programowanie obiektowe

## Wykład 2.

Marcin Młotkowski

1 marca 2024

# Plan wykładu

- 1 Mały wstęp
- 2 Klasy i obiekty
  - Deklaracja klas
  - Tworzenie obiektów
  - Wywoływanie metod
  - Tablice obiektów
- 3 Hermetyzacja w C#
- 4 Konstruktory
- 5 Kompilacja programów

# Plan wykładu

- 1 Mały wstęp
- 2 Klasy i obiekty
  - Deklaracja klas
  - Tworzenie obiektów
  - Wywoływanie metod
  - Tablice obiektów
- 3 Hermetyzacja w C#
- 4 Konstruktory
- 5 Kompilacja programów

# Krótki opis C#

- Obiektowy, z kontrolą typów;
- automatyczne odśmiecanie;
- standardy ISO i ECMA;
- podobny składniowo do C++;

Najnowsza wersja to 11.0

# C# czy C#?

# C# czy C<sup>#</sup>?

- C#: C-hash
- C<sup>#</sup>: "C krzyżyk" (cis)

# Plan wykładu

- 1 Mały wstęp
- 2 Klasy i obiekty
  - Deklaracja klas
  - Tworzenie obiektów
  - Wywoływanie metod
  - Tablice obiektów
- 3 Hermetyzacja w C#
- 4 Konstruktory
- 5 Kompilacja programów

## Przykładowe zadanie

Ewidencja pojazdów:

- co chcemy mieć w systemie:  
samochody osobowe, samochody ciężarowe, tramwaje;
- co chcemy wiedzieć:  
marka, rok produkcji, rejestracja;
- co chcemy robić:  
drukować informacje o danych.

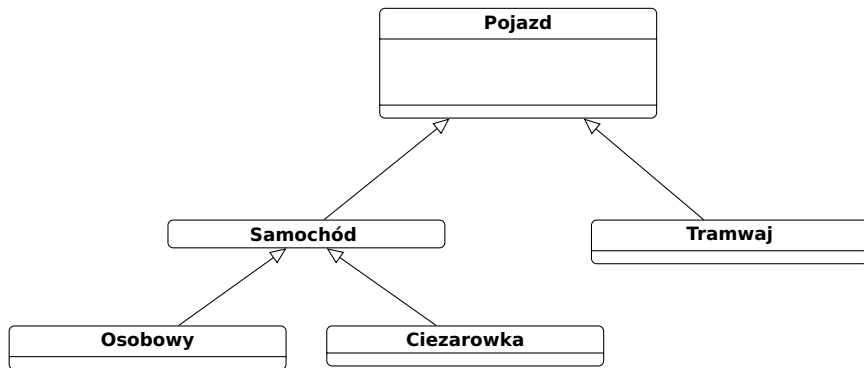


## Przykładowe zadanie

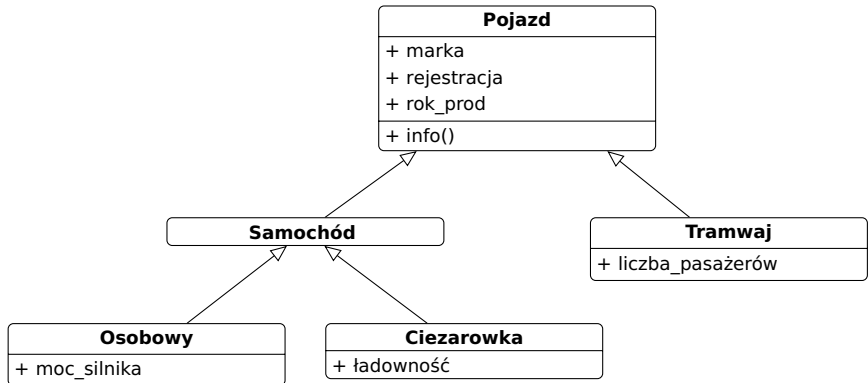
Ewidencja pojazdów:

- co chcemy mieć w systemie:  
**samochody osobowe, samochody ciężarowe, tramwaje;**
- co chcemy wiedzieć:  
marka, rok produkcji, rejestracja;
- co chcemy robić:  
drukować informacje o danych.

# Schemat zależności między obiektami rzeczywistymi



# Schemat zależności między obiektami rzeczywistymi



# Implementacja w C#

```
class Pojazd {  
    public string marka;  
    public string rejestracja;  
    public string rok_prod;  
    public void info() {  
        Console.WriteLine("Marka: {0}, rocznik: {1}}",  
            marka, rok_prod);  
    }  
}
```

# Deklaracja podklasy Samochod

```
class Samochod : Pojazd {  
    bool hybryda;  
    public void info() {  
        base.info();  
        Console.WriteLine("hybryda: {0}", hybryda);  
    }  
}
```

# Analiza przykładu

- Klasa Pojazd definiuje
  - pole marka
  - pole rejestracja
  - pole rok\_prod
  - metodę info()
- Klasa Samochód dziedziczy
  - pole marka
  - pole rejestracja
  - pole rok\_prod
- Klasa Samochód dodaje
  - pole hybryda
- Klasa Samochód definiuje na nowo
  - metodę info()

# Uzupełnienie 1.

## Dostęp do metody nadklasy

```
public void info() {  
    base.info();  
    Console.WriteLine("hybryda: {0}", hybryda);  
}
```

To jeszcze nie jest poprawny program.

## Uzupełnienie 2.

Rozwiązanie konfliktu

```
class Pojazd {  
    public void info() { ... }  
}  
class Samochod : Pojazd {  
    public void info() { ... }  
}
```



## Uzupełnienie 2.

Rozwiązanie konfliktu

```
class Pojazd {  
    virtual public void info() { ... }  
}  
class Samochod : Pojazd {  
    override public void info() { ... }  
}
```

## Uzupełnienie 3.

W większości języków jest dostępna domyślnie klasa `Object` będąca nadklasą (superklasą) wszystkich innych klas.

# Jak utworzyć obiekt

## Instrukcja new

```
Klasa var;  
var = new Klasa();
```

# Jak utworzyć obiekt

## Instrukcja new

```
Klasa var;  
var = new Klasa();
```

Do utworzenia obiektu konieczna jest klasa!

# Inne metody tworzenia obiektów

## Klonowanie obiektów

```
Klasa varprim;  
varprim = var.Clone();a
```

---

<sup>a</sup>Uwaga: małe oszustwo

# Inne metody tworzenia obiektów

## Klonowanie obiektów

```
Klasa varprim;  
varprim = var.Clone();a
```

---

<sup>a</sup>Uwaga: małe oszustwo

## Fabryki obiektów

```
Pojazd = FPM.construct("tramwaj");
```

Każde wywołanie metody jest związane z obiektem.

Każde wywołanie metody jest związane z obiektem.

```
Samochod limuzyna = new Samochod();  
limuzyna.info();
```



Każde wywołanie metody jest związane z obiektem.

```
Samochod limuzyna = new Samochod();  
limuzyna.info();
```

# Zagadka

Jak wywołać w metodzie własną metodę?

## Przykład

```
class Rekurencja {  
  
    public int gcd(int x, int y) {  
        if (x == 0) return y;  
        return ???gcd(y mod x, x);  
    }  
  
}
```

## Przykład

```
class Rekurencja {  
  
    public int gcd(int x, int y) {  
        if (x == 0) return y;  
        return this.gcd(y mod x, x);  
    }  
  
}
```

## Zmienna this (base)

Co warto wiedzieć o **this**:

- **this** jest zmienną obiektu wskazującą na obiekt;
- **this** można użyć tylko w metodach;
- **this** jest zawsze domyślnie zadeklarowane;
- nie można zmieniać wartości **this**.

## Inne zastosowania `this`

```
class Element {  
    public Element next;  
    public void set(Element e) {  
        this.next = e;  
        e.next = this;  
    }  
}
```

## Inne zastosowania **this**

```
class Element {  
    public Element next;  
    public void set(Element e) {  
        this.next = e;  
        e.next = this;  
    }  
}
```

```
Element a = new Element();  
Element b = new Element();  
Element c;  
a.set(b);  
c = a;  
while (c != null) c = c.next;
```

## Uwagi o this

- zwykle nie jest konieczne używanie **this**

```
return this.gcd(y mod x, x);
```



## Uwagi o this

- zwykle nie jest konieczne używanie **this**

```
return gcd(y mod x, x);
```

## Uwagi o this

- zwykle nie jest konieczne używanie **this**

```
return gcd(y mod x, x);
```

- this** może być konieczne do rozstrzygnięcia niejednoznaczności

```
int x;  
public void set(int x) {  
    this.x = x;  
}
```

# Tablice w C#

Tablice to też obiekty!

# Deklarowanie i inicjowanie tablic

## Deklaracja tablicy

```
Pojazd[ ] parking;
```

# Deklarowanie i inicjowanie tablic

## Deklaracja tablicy

```
Pojazd[ ] parking;
```

## Inicjowanie tablicy

```
parking = new Pojazd[2];
```

# Deklarowanie i inicjowanie tablic

## Deklaracja tablicy

```
Pojazd[ ] parking;
```

## Inicjowanie tablicy

```
parking = new Pojazd[2];
```

## Wszystko razem

```
Pojazd[ ] parking = new Pojazd[100];
```

Uwaga: wszystkie miejsca w tablicy mają wartość **null**!

## Przetwarzanie tablic

```
parking[0] = new Pojazd();  
parking[1] = new Samochod();  
  
foreach(Pojazd p in parking)  
    p.info();
```

# Plan wykładu

- 1 Mały wstęp
- 2 Klasy i obiekty
  - Deklaracja klas
  - Tworzenie obiektów
  - Wywoływanie metod
  - Tablice obiektów
- 3 Hermetyzacja w C#
- 4 Konstruktory
- 5 Kompilacja programów



## Domyślna widoczność pól i metod

Jeśli nie określono inaczej, pola i metody są niewidoczne z zewnątrz.

# Przykład

```
class Klasa {  
    string id;  
    void info() {  
        Console.WriteLine("Identyfikator: {0}", this.id);  
    }  
}
```

```
class Klasa {  
    string id;  
    void info() {  
        Console.WriteLine("Identyfikator: {0}", this.id);  
    }  
}
```

Dobrze

Klasa obiekt = **new** Klasa()

# Przykład

```
class Klasa {  
    string id;  
    void info() {  
        Console.WriteLine("Identyfikator: {0}", this.id);  
    }  
}
```

Dobrze

Klasa obiekt = **new** Klasa()

Źle

obj.id;  
obj.info();

```
class Klasa {  
    string id;  
    public void info() {  
        Console.WriteLine("Identyfikator: {0}", this.id);  
    }  
}
```

```
class Klasa {  
    string id;  
    public void info() {  
        Console.WriteLine("Identyfikator: {0}", this.id);  
    }  
}
```

Źle

obiekt.id

Dobrze

obiekt.info();

## Dziedziczenie a widoczność

```
class Klasa {  
    string id;  
    public void info() {  
        Console.WriteLine("Ident: {0}", this.id);  
    }  
}  
Klasa obj = new Klasa()
```

```
class Podklasa : Klasa {  
    public void myinfo() {  
        Console.WriteLine("Ident: {0}", this.id);  
    }  
}
```

## Widoczność w podklasie

```
class Klasa {  
    protected string id;  
    public void info() {  
        Console.WriteLine("Ident: {0}", this.id);  
    }  
}
```



## Podsumowanie

Poziomy dostępu do metod i pól

- **private** (domyślny) – tylko metody zdefiniowane w tej samej klasie;
- **protected** – widoczność w podklasie
- **public** – wszędzie.

# Plan wykładu

- 1 Mały wstęp
- 2 Klasy i obiekty
  - Deklaracja klas
  - Tworzenie obiektów
  - Wywoływanie metod
  - Tablice obiektów
- 3 Hermetyzacja w C#
- 4 Konstruktory
- 5 Kompilacja programów

## Stan początkowy obiektu

Ustalanie początkowych wartości zmiennych

- wartości domyślne ustalone w standardzie języka;
- przypisanie wartości w momencie utworzenia zmiennej;
- implementacja konstruktora.

## Wartości domyślne zmiennych

- `bool` – `false`;
- `string` – `""`;
- zmienna typu klasa – `null`

## Wartości domyślne zmiennych

- `bool` – `false`;
- `string` – `""`;
- zmienna typu klasa – `null`

### Uwaga

Można też przypisywać domyślne wartości funkcją `default`:  
`typ zmienna = default(typ);`

## Przypisanie wartości w miejscu deklaracji

```
int x = 12;  
Element e = new Element();
```

# Konstruktor

## Cechy konstruktora

- konstruktor to metoda, ale specjalna;
- jest to metoda wywoływana natychmiast po utworzeniu obiektu;
- konstruktor ma nazwę taką jak nazwa klasy;
- konstruktorów nie można jawnie wywoływać (prawie ;-);
- konstruktorów może być kilka.

# Deklaracja konstruktora

```
class Pojazd {  
    string marka;  
    int rok_prod;  
  
    public Pojazd() {  
        this.marka = "Syrena";  
        this.rok_prod = 1955;  
    }  
  
    public Pojazd(string marka) {  
        this.marka = marka;  
    }  
}
```



# Deklaracja konstruktora

```
class Pojazd {  
    string marka;  
    int rok_prod;  
  
    public Pojazd() {  
        this.marka = "Syrena";  
        this.rok_prod = 1955;  
    }  
  
    public Pojazd(string marka) {  
        this.marka = marka;  
    }  
}
```

## Użycie konstruktora

```
Pojazd p = new Pojazd();  
Pojazd w = new Pojazd("wehikuł czasu");
```

# Konstruktory i dziedziczenie

```
class Pojazd {  
    string marka = "";  
    public Pojazd(string marka) { this.marka = marka; }  
    public Pojazd() { this.marka = "syrena"; }  
}
```

```
class Samochod : Pojazd {  
    bool hybryda;  
    public Samochod(string marka, bool hybryda):  
        base(marka)  
    {  
        this.hybryda = hybryda;  
    }  
    public Samochod() {  
        /* Automatyczne wywołanie konstruktora  
        klasy Pojazd */  
    }  
}
```

# Destruktor

Destruktor (finalizator) to bezparametrowa metoda wywoływana przy usuwaniu obiektu z pamięci.

## Przykład

```
class Klasa {  
    ~Klasa() { ... }  
}
```

# Destruktor

Destruktor (finalizator) to bezparametrowa metoda wywoływana przy usuwaniu obiektu z pamięci.

## Przykład

```
class Klasa {  
    ~Klasa() { ... }  
}
```

## Uwaga

Nie wiadomo, kiedy obiekt będzie usunięty z pamięci, być może dopiero po zakończeniu programu.

# Plan wykładu

- 1 Mały wstęp
- 2 Klasy i obiekty
  - Deklaracja klas
  - Tworzenie obiektów
  - Wywoływanie metod
  - Tablice obiektów
- 3 Hermetyzacja w C#
- 4 Konstruktory
- 5 Kompilacja programów

# Początek programu

Początkiem programu jest publiczna statyczna metoda Main

## Przykład

```
class MojProgram {  
    public static void Main() {  
        ...  
    }  
}
```

## Rozszerzenia plików

Domyślnym rozszerzeniem pliku jest **\*.cs**

# Przykład programu

plik.cs

```
using System;

class Pojazd {
    ...
}

class Samochod : Pojazd {
    ...
}

class MojProgram {
    public static void Main() {
        Console.WriteLine("Hello world");
    }
}
```



# Środowiska: MS Windows

Microsoft Visual Studio

# Środowiska: MS Windows

Microsoft Visual Studio

CLI

Kompilator CLI:

```
c:\Moje Dokumenty> csc plik.cs
```

## Środowiska: Linux (macOS)

Mono

```
$ mcs plik.cs
```

IDE: monodevelop

## Środowiska: Linux (macOS)

### Mono

```
$ mcs plik.cs
```

IDE: monodevelop

### Inne

.NET Core SDK

IDE: VS Code