



Akademia C# 4

Od Podstaw





Michał Rzepczak



IV rok AiR na W-4

Junior Developer

w

ANI·XE

Email: michal.rzepczak@gmail.com



0 Agenda

- 🔍 Dziedziczenie
- 🔍 Hermetyzacja
- 🔍 Konstruktory a dziedziczenie?
- 🔍 Przeciążenia
- 🔍 Interfejsy
- 🔍 Klasy abstrakcyjne



1 Dziedziczenie

- 🔍 Korzystanie z funkcjonalności klasy bazowej i rozszerzanie ich
- 🔍 Można dziedziczyć tylko po jednej klasie
- 🔍 Klasa potomna może stać się bazową dla innych klas
- 🔍 Składnia:

```
public class FirstYearStudent : Student  
{  
}
```

Klasa potomna

Klasa bazowa

2 Hermetyzacja

Dostępne modyfikatory:

- **public** – widoczny wszędzie
- **protected** – widoczny dla klas dziedziczących
- **private** – widoczny tylko i wyłącznie dla klasy w której się znajduje
- **internal** – widoczny jak publiczny, jednak tylko w podzespole (program, biblioteka)
- **protected internal** – widoczny dla klas dziedziczących, tylko w podzespole




DEMO – Klasa Figure oraz Rectangle

3 A co z konstruktorami?

- **Konstruktory nie podlegają dziedziczeniu**
- **W każdej klasy musimy samemu stworzyć sobie konstruktor**
- **W celu uniknięcia powtarzalności kodu używa się inicjalizatorów – opcja wywołująca konstruktor klasy bazowej**

```
public Pochodna(string color) : base(color)
{
    Wywołanie konstruktora klasy bazowej
}
```



DEMO – Konstruktor klasy bazowej

Task 1.

- Stwórz klasę Ssak (ang. Mammal)
- Niech posiada publiczne właściwości: ilość kończyn i płeć + konstruktor
- Stwórz klasę Kot dziedziczącą po Ssaku, dodaj właściwość imię + konstruktor (wykorzystując konstruktor klasy bazowej)
- Stwórz metodę w klasie Kot wyświetlającą wszystkie informacje do których masz dostęp



4 Override - przeciążanie

- Rozszerzanie lub zmiana implementacji odziedziczonego elementu
- Dotyczy metod i właściwości wirtualnych
- Składnia:

W klasie bazowej:

```
public virtual void Hello()  
{  
    Console.WriteLine("Hello, I'm a student!");  
}
```

← Słowo kluczowe "virtual"

W klasie potomnej:

```
public override void Hello()  
{  
    Console.WriteLine("Hello, I'm a first year student!");  
}
```

← Słowo kluczowe "override"

DEMO — Przeciążenia

5

Interfejsy

- Definiuje właściwości, metody, zdarzenia
- Zawiera jedynie deklaracje składowych, nie ich definicje.
- Nie można stworzyć instancji interfejsu
- Zaleca się nazywanie interfejsów zaczynając od „I” np. `IMyInterface`
- Składnia:

```
public interface IMyInterface
{
    string Name { get; set; }
    void ShowSalary();
    double CalculateTax(double income);
}
```

6

Interfejsy w dziedziczeniu

- Klasa może dziedziczyć po wielu interfejsach
- Klasa bądź struktura, która dziedziczy po interfejsie musi implementować wszystkie jej metody w przeciwnym wypadku program się nie skompiluje

```
public class Rectangle : Figure, IFigures, IMathCalculations  
{
```

Jedna klasa

Wiele interfejsów



DEMO – Interfejsy a dziedziczenie

7 Klasy abstrakcyjne

- ❶ Nie można stworzyć instancji klasy abstrakcyjnej
- ❷ Spełnia swój cel jedynie w hierarchii dziedziczenia (jako klasa bazowa)
- ❸ Jeśli implementowaliśmy interfejs musieliśmy użyć w klasie wszystkie jego metody, w przypadku klasy abstrakcyjnej nie ma takiej konieczności – muszą pojawić się jedynie definicje metod abstrakcyjnych
- ❹ Metody abstrakcyjne działają podobnie do metod wirtualnych – ale nie posiadają ciała

DEMO – Klasy abstrakcyjne

Task 2.

- Przerób klasę Ssak na abstrakcyjną
- Dodaj metodę abstrakcyjną wypisującą na ekran konsoli dźwięki wydawane przez tego ssaka
- Odpowiednio nadpisz metodę w klasie potomnej
- Przetestuj działanie metody



Zadanie domowe 1. - Figury

- Zdefiniuj abstrakcyjną klasę **Figura**, która określi wymagania dla klas po niej dziedziczących. W klasie ma być publiczna właściwość typu **string** „**Name**” (nazwa figury np.: „kwadrat”, „figura”), metoda wyświetlająca nazwę figury (np.: „To jest kwadrat”), deklaracja metody obliczającej pole figury oraz deklaracja metody obliczającej jej obwód.
- Stwórz klasy dziedziczące po **Figura**: **Koło**, **Prostokąt**, **Kwadrat** oraz dla chętnych **Trapez**. W nich stwórz prywatne pola, które potrzebne będą do przeciążanych metod.
- Pamiętaj o utworzeniu odpowiednio sparametryzowanych konstruktorów pobierających jako parametr niezbędne dane do wykonania obliczeń (np.: długość boku itd.) i właściwość „**Name**”.
- Nadpisz (przeciąż — słowo kluczowe **override**) metody obliczające pole i obwód dla każdej figury
- W klasie **Program** (czyli tam, gdzie metoda **Main**) stwórz odpowiednie metody prezentujące działanie poszczególnych figur i wywołaj je w metodzie **Main**



Pytania?



Dziękuję za uwagę!