Programowanie komputerów.
Ćwiczenia 6.

Dziedziczenie

dziedziczenie.cs

Metody Wirtualne

Metoda oznaczona jako wirtualna (**`virtual`**) może być nadpisana (**`override`**) w klasach pochodnych, w których ją zdefiniowano.

wirtualna.cs

Klasa Abstrakcyjna

```
public abstract class Figura { ... }
```

Klasa abstrakcyjna jest zdefiniowana jedynie na potrzeby dziedziczenia. Nie można utworzyć obiektu na jej podstawie.

abstrakcyjna_klasa.cs

Metoda Abstrakcyjna

Za pomocą słowa kluczowego **`abstract`** można także tworzyć metody abstrakcyjne. Taka metoda nie ma własnej implementacji. Implementacją muszą się zająć klasy pochodne (które dziedziczą klasę, jaka zawiera daną metodę abstrakcyjną).

Metoda abstrakcyjna może być utworzona wyłącznie w klasie abstrakcyjnej.

abstrakcyjna_klasa_metoda.cs

STRUKTURA

```
[Modyfikatory] struct MojaStruktura
{
      // Ciało struktury
}
```

Struktura a Klasa:

- Różnica w składni – dla struktur używa się słowa kluczowego **`struct`** (dla klas **`class`**)
- Struktura jest typem wartościowym (klasa jest typem referencyjnym)
- Struktura nie może dziedziczyć po klasie, ani być przedmiotem dziedziczenia (ale może implementować interfejsy)
- W strukturze nie można zadeklarować konstruktora bez argumentów
- Składowe struktury nie mogą być inicjalizowane w momencie deklarowania

Pamięć w środowisku .NET dzielona jest na dwie części:

- Stos (stack) jest strukturą danych używanych do przechowywania zmiennych typu wartości. Np. gdy tworzymy zmienną typu int, to jej wartość zapisuje się na stosie. Na stosie zapisywane są także informacje o każdym wywołaniu metody i są z niego usuwane w chwili zaskoczenia działania metody.
- Sterta (heap) wykorzystywana jest do przechowywania zmiennych/obiektów typu referencyjnego. W momencie tworzenia egzemplarza klasy na stercie rezerwuje się miejsce na obiekt, a adres do niego umieszczany jest na stosie.

- Rezerwacja i zwalnianie pamięci na stosie jest zdecydowanie szybsze od tych samych operacji wykonywanych na stercie. Dlatego dla przechowywania danych o niewielkich rozmiarach lepiej jest wykorzystać typy wartości, a dla danych o dużych rozmiarach lepiej użyć typów referencji.

Informacje na temat organizacji pamięci (stosu i sterty) znajdują się w podręczniku „Wstęp do programowania w C#" na stronie http://c-sharp.ue.katowice.pl/ w podrozdziałach ***2.1.1 Typ wartościowy i typ referencyjny*** oraz ***6.4 Typ referencyjny w kolejnej odsłonie***

Struktura czy Klasa?

- Przy założeniu, że nie potrzebujemy dziedziczenia kierujemy się w wyborze (użyć klasy czy struktury) kwestią pamięci.
- Ponieważ Struktura jest typem wartości a Klasa jest typem referencji stosujemy Klasę gdy ma ona obsługiwać duże zbiory danych, a strukturę gdy ma pracować na niewielkich zbiorach.
- Ważna jest częstość wywołań metod mających jako argument obiekt(y) – jeśli duża to lepiej klasa.

TYP WYLICZENIOWY (ENUM)

Typ wyliczeniowy - **enum** - to specjalny typ wartościowy służący do definiowania grupy stałych wartości liczbowych o określonych nazwach, np. :

```
public enum Kierunek
{
    prawo,
    lewo,
    gora,
    dol
}
// . . .
// przykład wykorzystania typu wyliczeniowego Kierunek:
Kierunek k1 = Kierunek.gora;
```

Każda składowa typu wyliczeniowego posiada skojarzoną ze sobą wartość typu całkowitego. Domyślnie wartości liczbowe składowych mają typ **int**, a deklaracjom stałych w typie wyliczeniowym automatycznie przypisywane są kolejne wartości liczbowe (0, 1, 2 itd.)

Można określić alternatywny typ wartości liczbowych (np. **byte**). Można też samodzielnie określać wartości liczbowe kojarzone z poszczególnymi składowymi typu wyliczeniowego (w dowolnym porządku), np.:

```
public enum Kierunek: byte
{
    prawo = 1,
    lewo = 2,
    gora = 10,
    dol = 11
}
```

Przykładowe wbudowane typy wyliczeniowe

ConsoleColor z wartościami: Black, Blue, Cyan, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, DarkYellow, Gray, Green, Magenta, Red, White, Yellow, np.

```
Console.SetCursorPosition(5, 2);  // (kolumna, wiersz)
Console.ForegroundColor = ConsoleColor.Red;
Console.Write("Komunikat w kolorze czerwonym");
```

ConsoleKey z wartościami: Delete, Enter, Escape, RightArrow, LeftArrow, Insert, Home, F1, F2,... (i wiele innych), np.

```
Console.WriteLine("Naciśnij Esc, jeśli chcesz dalej");
if (Console.ReadKey(true).Key == ConsoleKey.Escape)
    Console.WriteLine("Dalej");
```

ZADANIA

Zadanie 1.

Napisz program, który pobierze dwie liczby od użytkownika i podzieli pierwszą liczbę przez drugą. Program powinien zweryfikować czy można zamienić liczby na typ numeryczny (int) oraz sprawdzi czy druga liczba jest różna od zera.

Dodatkowo przyjmuj zasadę, że pierwsza liczba nie może być równa 10.

Zadanie 2.

Napisz program z użyciem struktury **`Zespolone`**.

Struktura ta ma posiadać:

- część rzeczywistą i część urojoną
- konstruktor inicjalizujący część rzeczywistą i urojoną
- metodę `ToString` nadpisującą metodę z klasy `System.Object` , która wyświetla liczbę zespoloną w formacie (`a+bi`)
- metodę `Equals` nadpisującą metodę z klasy `System.Object` sprawdzającą, czy dwie liczby zespolone są równe (przewodnik https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/how-to-define-value-equality-for-a-type)
- przeładowany operator`+` dodający dwie liczby zespolone

Zadanie domowe 1.

Napisz program z klasą bazową **Prostokat** i klasą pochodną **Blat**. W klasie bazowej zdefiniuj metodę obliczającą powierzchnię prostokąta i odpowiednie pola. W klasie **Blat** należy zdefiniować metodę obliczającą koszt blatu uwzględniając rodzaj blatu (blat granitowy 600 zł/m2, blat drewniany 250 zł/m2). Wyposaż obie klasy w metodę zwracającą tekst opisujący dane **ToString**. W metodzie **Main** zdefiniuj kilka obiektów dla obu klas i wypisz dane o tych obiektach.

Zadanie domowe 2.

Napisz program z użyciem trzech klas. Bazowa klasa abstrakcyjna **Figura** z polem opisującym kolor figury (z użyciem typu **ConsoleColor**) i abstrakcyjną metodą **Pole**. Klasa pochodna **Kolo** ma zapamiętywać promień koła. Natomiast klasa pochodna **Prostokat** ma zapamiętywać długości obu boków. W klasach pochodnych wykonaj implementacje metody obliczającej powierzchnię figur. Wyposaż wszystkie wymienione klasy w metodę zwracającą tekst opisujący dane **ToString**. W metodzie **Main** zdefiniuj tablicę figur (jedną wspólną dla kół i prostokątów) i zaprezentuj dane o figurach.

Zadanie domowe 3.

Napisz program stanowiący (po pewnych modyfikacjach) połącznie dwóch poprzednich z następującym schematem dziedziczenia:

Figura -> **Kolo** oraz
Figura -> **Prostokat** -> **Blat**.

(*Znaki strzałek używane są w dokumentacji języka C# przy opisie zależności wynikających z dziedziczenia. Grot strzałki wskazuje na klasę pochodną*).

Zadanie domowe 4.

Napisz program, w którym będą zaimplementowane dwie klasy: **Szescian** i dziedzicząca po niej klasa **Prostopadloscian**. W klasie **Szescian** powinny być chronione (`protected`) pola: **nazwa** i **krawedz** (`double`) oraz wirtualna metoda obliczająca objętość sześcianu. W klasie pochodnej **Prostopadloscian** zdefiniuj odpowiednio konstruktor, dodaj pola dla dwóch pozostałych krawędzi. Zdefiniuj nadpisanie metody obliczającej objętość bryły. W obu klasach zdefiniuj metodę **ToString** zwracającą łańcuch znakowy z informacjami o bryle (łącznie z objętością). W metodzie **Main** zdefiniuj tablicę brył (kilka sześcianów i kilka prostopadłościanów w jednej tablicy). Wyświetl informacje o bryłach wykorzystując pętlę.

Zadanie domowe 5.

Napisz program z abstrakcyjną klasą **Macierz** posiadającą abstrakcyjną metodę **Wyznacznik**. Po klasie tej ma dziedziczyć klasa **MacierzTrojkatna** posiadająca nadpisanie metody **Wyznacznik** (wyznacznik macierzy trójkątnej oblicza się jako iloczyn elementów na głównej przekątnej). Ponadto po klasie **Macierz** ma dziedziczyć także klasa **MacierzJednostkowa** z

nadpisaną metodą **`Wyznacznik`** (wyznacznik macierzy jednostkowej jest równy 1). Wymienione klasy mają posiadać także inne składowe (w tym pola) potrzebne do opisania macierzy i wyświetlania jej zawartości. W metodzie **`Main`** stwórz tablicę dla obiektów obu klas nieabstrakcyjnych (**`MacierzTrojkatna`** i **`MacierzJednostkowa`**) i wyświetl informacje o tych obiektach na konsoli.

Zadanie domowe 6.

Do programu z poprzedniego zadania dodaj klasę **`MacierzDiagonalna`** (dziedziczącą po klasie **`MacierzTrojkatna`**). Macierz diagonalna to macierz, której wszystkie współczynniki leżące poza główną przekątną są zerowe (jest to macierz górno- i dolnotrójkątna jednocześnie). Wyznacznik macierzy diagonalnej oblicza się jako iloczyn elementów na głównej przekątnej. Istnieje także szczególny przypadek macierzy diagonalnej – macierz skalarna. Jest to macierz, której wszystkie elementy na głównej przekątnej są równe. Wyznacznik takiej macierzy oblicza się jako c^n (wartość elementu występujący na przekątnej do potęgi n). Dodaj do programu klasę dla macierzy skalarnej umieszczając ją odpowiednio w diagramie dziedziczenia. Dodatkowo klasę **`MacierzSkalarna`** należy zdefiniować jako zamkniętą, wskazując, że dalsze dziedziczenie po niej nie jest możliwe (**`sealed`**).