Programowanie komputerów.
Ćwiczenia 5.

WYJĄTKI

```
try
{
    // blok kodu w którym może wystąpić wyjątek
}
catch (TypWyjatku_1 blad)
{
    // obsług wyjątku dla TypWyjatku_1
}
catch (TypWyjatku_2 blad)
{
    // obsług wyjątku dla TypWyjatku_2
}
finally
{
    // blok kodu, który wykona się zawsze
}
```

Wyjątki mogą być także utworzone przez programistę za pomocą słowa kluczowego **`throw`**:

```
if mianownik == 0
{
    Throw new ArgumentException(„Zero w mianowniku”)
}
```

PRZEŁADOWANE OPERATORY

W języku C# można przeładować (przeciążać) zarówno metody jak i operatory. Przeciążenie operatora dokonuje się poprzez wpisanie słowa kluczowego operator zamiast nazwy metody i obok wpisać należy sam operator (np. +, -, * lub inny). Przeładowany operator musi być publiczny i statyczny. Przynajmniej jeden argument musi być tego samego typu, w którym zadeklarowano dany operator.

Operatory można przeciążać w celu zapewnienia bardziej naturalnej składni dla typów własnych programisty. Technikę przeciążania operatorów najlepiej jest stosować przy implementacji własnych struktur reprezentujących dość proste typy danych.

Na stronie https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/operator-overloading umieszczona jest lista operatorów, które można przeciążyć oraz wykaz tych, których nie można.

Operatory równości i porównywania

Kompilator C# wymaga definicji obu operatorów stanowiących logiczne pary:

`==  i  !=`

`<    i    >`

`<=  i  =>`

W większości przypadków przeciążenie operatorów == i != pociąga za sobą konieczność dodatkowego przeciążenia metod **`Equals`** i **`GetHashCode`** zdefiniowanych w klasie **`System`**.**`Object`**. W przypadku ich braku kompilator zgłosi ostrzeżenie.

Przykładowa definicja operatora +

```
public static double operator +(Towar t1, Towar t2)
{
    return (t1.cena + t2.cena);
}
```

Tak zdefiniowany operator zwróci sumę cen obu towarów. Wykorzystanie operatora może wyglądać następująco:

```
Towar t1 = new Towar("ołówek", 2.5);
Towar t2 = new Towar("długopis", 3.2);
Console.WriteLine(t1+t2);   // wypisze 5,7
```

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.

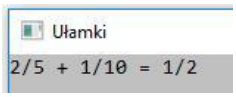
Napisz program z użyciem struktury **`Zespolone`**.

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 za pomocą struktury **`Ułamki`**, który dodaje ułamki zwykłe.



W programie wykorzystaj operatory przeciążone dla dodawania, odejmowania i mnożenia dwóch ułamków, a ponadto zdefiniuj operator mnożenia ułamka przez liczbę całkowitą.

Zadanie domowe 2.

Napisz program z użyciem struktury **`Pogoda`**, który będzie służył do ewidencji pomiarów warunków pogodowych w danej stacji pogodowej. Struktura ta powinna posiadać takie pola jak czas pomiaru, temperatura, prędkość wiatru i jego kierunek, wilgotność oraz zachmurzenie (duże, średnie, małe, brak). Zarówno kierunek wiatru jak i zachmurzenie opisz przy pomocy typu wyliczeniowego. W metodzie **`Main`** utwórz tablicę obiektów z przykładowymi pomiarami i wyświetl wykaz pomiarów w formie tabelarycznej.

Zadanie domowe 3.

Napisz program z użyciem struktury **`KandydatNaStudia`**, która ma posiadać następujące pola: **`nazwiskoKandydata`**, **`imieKandydata`**, **`punktyMatematyka`**, **`punktyInformatyka`**, **`punktyJezykObcy`**. W trzech ostatnich polach mają być zapisane punkty za przedmioty zdawane na maturze odpowiednio z przedmiotów: matematyka, informatyka i język obcy (dla uproszczenia uwzględniamy tylko jeden poziom zdawanej matury, np. podstawowy). Jeden punkt to jeden procent (tj. kandydat, który ma 55% z matematyki otrzyma `55` punktów z tego przedmiotu). Struktura ma posiadać metodę obliczającą łączną liczbę punktów kandydata według przelicznika: 0,6 punktów z matematyki `+ 0,5` punktów z informatyki `+ 0,2` punktów z języka obcego. W metodzie **`Main`** utwórz obiekty dla struktury (jako elementy tablicy) dla kilku kandydatów i pokaż listę kandydatów, zawierającą nazwisko i obok, w tej samej linii, obliczoną łączną liczbę punktów.

Zadanie domowe 4.

Napisz program z klasą opisującą macierze **`Macierz`**. Klasa ta ma posiadać przeładowany operator `+`, który realizuje dodawanie dwóch macierzy oraz przeładowany operator `*`, który realizuje mnożenie dwóch macierzy. Ponadto w klasie ma być nadpisana metoda **ToString**, która zwraca string z elementami macierzy (każdy wiersz w osobnej linii).

Zadanie domowe 5.

W języku C# jest dostępna struktura **DateTime**, która posiada kilka przeciążonych operatorów. Wykonaj swoją strukturę dla czasu, która ma posiadać konstruktor inicjalizujący czas za pomocą dwóch argumentów - jeden dla godziny (0-23) i drugi dla minut (0-59). W programie należy zdefiniować dwa przeciążone operatory `+` oraz `−`, które pozwolą dodawać i odejmować podane czasy. Oba operatory mają zwrócić obiekt tej struktury. W strukturze zdefiniuj także nadpisaną

metodę **`ToString`**, która ma zwracać łańcuch znakowy opisujący czas w formacie H:mm (np. 21:55). Przykładowe wykorzystanie struktury:

```
Czas t1 = new Czas(23,30);
Czas t2 = new Czas(1,0);
Console.WriteLine(t1 + t2);   //  0:30
Console.WriteLine(t1 – t1);   // 22:30
```