TYPY GENERYCZNE

Typy generyczne umożliwiają zbudowanie klas, które działają na jeszcze nie zdefiniowanym typie. Dzięki temu możemy przygotować klasy, których metody operują na specyficznym typie i umożliwiają statyczne sprawdzenia poprawności typów, czego nie można by osiągnąć operując na typie Object.

Zadanie 1

Utworzyć funkcje generyczne:

- *void WypiszObiekt<T>(T obiekt)* - wypisuje obiekt do konsoli
- *bool PierwszyWiekszy<T>(T o1, T o2)* - zwraca true, jeśli obiekt o1 jest większy (typ T musi umożliwiać porównanie)

Zadanie 2

Zaimplementować dziedziczące po sobie klasy:

- Sportowiec (pola typu string: Imię, Nazwisko)
- KierowcaFormula1 (dodatkowo pole typu string: NazwaDrużyny)
- RekorKierowcy (dodatkowo pole typu int: RekordZyciowy)

Pola mają być prywatne, wypełniane w konstruktorach.

Każda klasa ma mieć nadpisaną metodę ToString umożliwiającą wypisanie wszystkich informacji.

Zadanie 3

Testowanie poprawności zadania 2:

Utworzyć listę obiektów typu sportowiec, dodać do niej obiekty wszystkich trzech wcześniej zdefiniowanych klas. Wywołać w pętli metodę ToString dla każdego obiektu.

Przewodnik programowania w języku C#

https://docs.microsoft.com/pl-pl/dotnet/csharp/programming-guide/generics/

DEKLARACJA

Przy deklaracji klasy wszystkie jej elementy muszą mieć sprecyzowany typ, aby móc go nie określać przy deklaracji. Konieczne jest nadanie mu jakiegoś aliasu, który to w deklaracji klasy będzie wykorzystywany zamiast typu. Alias ten jest zapisywany w nawiasach trójkątnych <>.

```
public class CSharpBox<T>
{
    public T box;
}
```

gdzie pole box będzie miało typ zgodny z parametrem generycznym.

```
CSharpBox<int> mbox = new CSharpBox<int>();
CSharpBox<string> mbox2 = new CSharpBox<string>();

mbox.box = 5;
mbox2.box = "5";
```

Różnorodność typów obsługiwanych przez nasz typ generyczny może być ograniczona. W tym celu po deklaracji naszej klasy wykorzystujemy słowo kluczowe **where** aliasTypu :

- class – typ musi być typem referencyjnym
- struct – typ musi być typem wartości
- new() – typ musi posiadać publiczny konstruktor bez parametrów. To ograniczenie musi być stosowane na samym końcu listy ograniczeń
- nazwaTypuBazowego lub NazwaInterfejsu – parametr musi dziedziczyć po klasie bazowej lub implementować interfejs

```
public class CSharpDateWithError<T> where T: class
{
    public T SortedObject {get; set;}
    public Error {get; set;}
}
```

METODY

Metoda generyczna, która może przyjąć parametry o dowolnym typie kreślonym wcześnie w nawiasach:

```
static string JoinString<T>(T a, T b)
{
    return a.ToString() + b.ToString();
}
```

Może również zwracać typ generyczny:

```
static T Return<T>(T a, T b)
{
    if (a.Date() > b.Date())
        return a;
    else
```

```
        return b;
}
```

Kolejny przykład metody generycznej, która wydrukuje wszystkie elementy tablicy typu T:

```
public static void Print<T>(T[] array)
{
    for (int i = 0; i <  array.Length; i++)
    {
        Console.WriteLine(array[i]);
    }
}
```

WYWOŁANIE generyczne metody

Przypadek metod generycznych nie znajdujących się w klasach generycznych:

```
public class NotGenericBox
{
    public static void Print<T>(T[] array)
    {
        for (int i = 0; i <  array.Length; i++)
        {
            Console.WriteLine(array[i]);
        }
    }
}

int[] NewArray = new []{ 1, 2, 3, 4 };
NotGenericBox.Print(NewArray);
```

DZIEDZICZENIE po klasach generycznych

```
class Base { }
class BaseG<T> { }
class G1<T> : Base { }
class G2<T> : BaseG<int> { }
class G3<T> : BaseG<T> { }
```

WARTOŚĆ DOMYŚLNA

C# oferuje słowo kluczowe “default”, które jest przydatne przy typach generycznych. Zwróci on wartość domyślną określonego typu.

```
static void Value<T>(ref T a)
{
    a = default(T);
}
```