TECHNOLOGIA

STAŁE WYLICZENIOWE JAKO OBIEKTY W C#

dr inż. Cezary Siwoń

magazyn.wsei.edu.pl

Wstęp

Język C# zawiera wiele elementów i koncepcji powstałych wcześniej języków obiektowych: C++, Java, Object Pascal. Najczęściej postrzegany jest jako odpowiednik języka Java, zarówno ze względu na przenośność kodu wykonywalnego jak i podobną składnię. Oczywiste jest, że język C# w wielu aspektach różni się od Javy. Większość różnic i nowych elementów C# daje znaczną przewagę nad Javą, która rozwija się w bardziej zachowawczy sposób. Jednym z nielicznych elementów Javy, który jest bardziej zaawansowany od odpowiednika w języku C# jest typ wyliczeniowy deklarowany słowem kluczowym enum.
W języku Java typ wyliczeniowy jest specjalnym rodzajem klasy, z której nie można wyprowadzać innych klas, ale można implementować interfejsy.
Stałe wyliczeniowe typu enum są instancjami zdefiniowanej klasy. Każda stała może posiadać pola i metody. Dokumentacja języka Java podaje poniższy przykład rozbudowanego wyliczenia:

```
public enum Planet {
   MERCURY (3.303e+23, 2.4397e6),
   VENUS   (4.869e+24, 6.0518e6),
   EARTH   (5.976e+24, 6.37814e6),
   MARS    (6.421e+23, 3.3972e6),
   JUPITER (1.9e+27,  7.1492e7),
   SATURN (5.688e+26, 6.0268e7),
   URANUS (8.686e+25, 2.5559e7),
   NEPTUNE (1.024e+26, 2.4746e7);
   private final double mass;  //
in kilograms
   private final double radius; //
in meters
   Planet(double mass, double
radius) {
       this.mass = mass;
       this.radius = radius;
   }
   private double mass() { return
mass; }
   private double radius() { return
radius; }
   // universal gravitational
constant (m3 kg-1 s-2)
   public static final double G =
6.67300E-11;
   double surfaceGravity() {
       return G * mass / (radius *
radius);
   }
   double surfaceWeight(double
otherMass) {
       return otherMass *
surfaceGravity();
   }
}
```

Celem artykułu jest przedstawienie możliwości tworzenie wyliczeń w języku C# o podobnych możliwościach, aby stałe były pełnymi obiektami, a wyliczenie było silnie typowane.

Typ enum w C#

Implementacja typu wyliczeniowego w języku C# jest zbliżona do języka C++. Każda stała posiada wartość całkowitą, która może być inicjowana automatycznie przez kompilator lub jawnie przypisana. Stałe o tej samej wartości całkowitej różnych typów nie są sobie równe. Przykład definicji typu Planet:

```
public enum Planet{
 MERCURY = 1,
    VENUS,
    EARTH,
    MARS,
    JUPITER,
    SATURN,
    URANUS,
    NEPTUNE
}
```

Taka definicja prowadzi do użycia instrukcji switch, aby podać masę lub promień danej planety.

```
switch (planet)
{
   case Planet.MARS:
       Console.WriteLine("Mars mass: 6.421e+23");
       break;
   case Planet.MERCURY:
       Console.WriteLine("Mercury mass: 3.303e+23");
       break;
   //pozostałe przypadki
}
```

Można także stworzyć słownik z kluczem w postaci stałej wyliczeniowej i wartością, która jest obiektem z właściwościami Radius i Mass. Są to jednak rozwiązania doraźne.
Lepszym rozwiązaniem jest zastosowanie metod rozszerzeń. Pozwala ono dodać metody do stałych wyliczeniowych (a w przyszłości możliwe będzie dodawanie także właściwości), co sprawia, że w kodzie klienckim stałe wyliczeń postrzegane są jak obiekty.

```
public static class PlanetExtensions
{

   public static readonly double G = 6.67300E-11;
   public sealed class PlanetDetails
   {
       public required double Mass { get; init; }
       public required double Radius { get; init; }
   }

   private static readonly Dictionary<Planet, PlanetDetails> _details =
new()
   {
       {
           Planet.MARS,
           new PlanetDetails()
           {
               Mass = 6.421e+23,
               Radius = 3.3972e6
           }
       },
       {
           Planet.MERCURY,
           new PlanetDetails()
           {
               Mass = 3.303e+23,
               Radius = 2.4397e6
           }
       },
 // pozostałe planety
   };
   public static PlanetDetails Details(this Planet planet)
   {
       return _details[planet];
```

```
}

 public static double SurfaceGravity(this Planet planet) {
 return G * planet.Details().Mass / (planet.Details().Radius *
planet.Details().Radius);
 }

 double static double SurfaceWeight(this Planet planet, double otherMass)
{
 return otherMass * planet.SurfaceGravity();
 }
}
```

Dane planet są dostępne w obiektach typu *PlanetDetails*. Obiekty te zwracane są metodą rozszerzenia *Details*. Metody *SurfaceGravity* i *SurfaceWeight* są wywoływane bezpośrednio na stałych.
Przedstawione rozwiązanie jest akceptowalne do tworzenia typów charakterystycznych dla tworzonej aplikacji, ale nie nadaje się do tworzenia bibliotek. Klasy rozszerzeń są importowane osobno, więc można stworzyć alternatywną klasę rozszerzeń i zmienić kontrakt stałych wyliczeniowych w kodzie klienckim. Pewnym rozwiązaniem może być umieszczenie w jednej przestrzeni nazw stałych i klasy rozszerzeń, co uniemożliwiałoby zaimportowanie innego rozszerzenia z tymi samymi składowymi. Nie zmienia to faktu, że takie rozwiązanie jest mało obiektowe: typ wyliczeniowy i składowe stałych wyliczeniowych są definiowane osobno, więc łatwo o błędy.

Własny typ wyliczeń
Najlepszym rozwiązaniem jest zdefiniowanie własnego typu, imitującego typ wyliczeniowy. Prowadzi to do silnie typowanych wyliczeń i zamknięcie wszystkich elementów w jednej klasie. Wymaga to większej pracy, jeśli chcemy zapewnić funkcje dostępne w natywnym typie wyliczeniowym: lista stałych, lista nazw, parsowanie itd.
Poniższa klasa jest prostym przykładem własnej klasy bazowej imitującej typ wyliczeniowy.

```
public abstract class ObjectEnum<T> where T: ObjectEnum<T>
{

   private static Dictionary<String, ObjectEnum<T>> _values;
   public static IEnumerable<string> Names => _values.OrderBy(e =>
e.Value._ordinal).Select(e => e.Key).AsEnumerable();
   public static IEnumerable<ObjectEnum<T>> Values =>
_values.Values.OrderBy(e => e._ordinal).AsEnumerable();

   private static int _ordinalCounter;
   private readonly string _name;
   private readonly int _ordinal = 0;
   public string Name => _name;
   public int Ordinal => _ordinal;

   static ObjectEnum()
   {
       _ordinalCounter = 0;
       _values = new ();
   }

   protected ObjectEnum(string name)
   {
       _ordinal = ++_ordinalCounter;
       _name = name;
       _values.Add(_name, this);
   }

   public static T? Parse(string name)
   {
       return _values.TryGetValue(name, out var v) ? v as T : throw new
ArgumentException("Invalid enum name!");
   }

   public static bool TryParse(string name, out T? value)
   {
       if (_values.TryGetValue(name, out var val))
       {
           value = val as T;
           return true;
       }

       value = null;
       return false;
   }

 public override string ToString()
 {
 return _name;
 }

 public static implicit operator
int(ObjectEnum<T> item)
 {
 return item.Ordinal;
 }
}
```

Parametr generyczny reprezentuje typ stałej wyliczeniowej.
Składowa statyczna *_values* jest słownikiem, który przechowuje pod nazwą stałej obiekt tej stałej.
Właściwość statyczna *Names* zwraca ciąg nazw stałych, posortowanych wg liczby porządkowej.
Właściwość statyczna *Values* zwraca ciąg stałych wyliczeniowych, posortowanych wg liczby porządkowej.
Właściwość instancyjna *Name* i *Ordinal* zwraca odpowiednio nazwę stałej i jej liczbę porządkową. Każdej właściwości odpowiada składowa prywatna: *_name* i *_ordinal*.
Składowa statyczna *_ordinalCounter*, jest licznikiem instancji i służy do nadawania liczb porządkowych każdej stałej.
Blok statyczny inicjuje licznik instancji i słownik *_values*.
Konstruktor inicjuje nazwę stałej na podstawie parametru oraz nadaje numer porządkowy i dodaje aktualnie utworzoną instancję do słownika *_values*.
Pozostałe metody realizują:

- *Parse* – na podstawie łańcucha wejściowego zwraca stałą wyliczeniową ze słownika, a w przypadku braku stałej o podanej nazwie zgłasza wyjątek.
- *TryParse* – wykonuje to samo co metoda *Parse*, ale zwraca wartość logiczną, a stała jest zwracana przez parametr wyjściowy.
- *ToString* – zwraca nazwę stałej
- Operator niejawnego rzutowania zwraca liczbę porządkową stałej.

Zdefiniowanie typu wyliczeniowego *Planet* polega na zdefiniowaniu klasy dziedziczącej po przedstawionym typie bazowym. Wewnątrz typu należy umieścić stałe, które są instancjami definiowanego typu. Dodać można także dowolne składowe, czyli w naszym przykładzie są to właściwości Mass i Radius oraz metody *SurfaceGravity* i *SurfaceWeight*.

Poniżej przykład klasy opisującej planety:

```
public sealed class PlanetEnum:
ObjectEnum<PlanetEnum>
{
   public static readonly PlanetEnum
Mars = new(nameof(Mars), 6.421e+23 ,
3.3972e6);
   public static readonly
```

```
PlanetEnum Mercury = new(nameof(Mercury), 3.303e+23, 2.4397e6);
 // pozostałe stałe

 public static readonly double G = 6.67300E-11;

 public double Mass { get; }
 public double Radius { get; }

 private PlanetEnum(string name, double mass, double radius) :
base(name)
 {
 Mass = mass;
 Radius = radius;
 }

 public double SurfaceGravity() {
 return G * Mass / Radius * Radius;
 }

 public double SurfaceWeight(double otherMass) {
 return otherMass * SurfaceGravity();
 }
}
```

Konstruktor jest prywatny, aby zablokować tworzenie instancji poza klasą. Konstruktor wywołuje konstruktor bazowy, który dodaje tworzoną instancję do listy wartości. Dodatkowo klasa jest typu sealed, czyli nie można po niej dziedziczyć.
Program demonstrujący zdefiniowany typ:

```
public static void Main(String[] args) {
   if (args.Length != 1) {
       Console.WriteLine("Usage: Planet <earth_weight>");
       Environment.Exit(1);
   }
   double earthWeight = double.Parse(args[0]);
   double mass = earthWeight/PlanetEnum.Earth.SurfaceGravity();
   foreach (var objectEnum in PlanetEnum.Values)
   {
       var p = (PlanetEnum)objectEnum;
       Console.WriteLine($"Your weight on {p} is
{p.SurfaceWeight(mass)}");
   }
}
```

Uruchomienie przykładu z argumentem 125 daje w efekcie poniższy wydruk:

```
Your weight on Mars is 13,43080655957162
Your weight on Mercury is 6,908885542168674
Your weight on Earth is 124,99999999999999
```

SmartEnum

Prezentowany przykład własnego typu wyliczeniowego nie zapewnia współpracy z wieloma bibliotekami i frameworkami. Można oczywiście uzupełnić go o dodatkowe klasy np. dokonujące serializacji czy współpracujące z EntityFramework.
Na szczęście powstała ogólnodostępna biblioteka SmartEnum, która znacznie ułatwia tworzenie wyliczeń zapewniając wsparcie dla wielu produktów z ekosystemu.NET. Jest dobrze przetestowana i przygotowana przez doświadczonych programistów.
Podstawowe klasy znajdują się w paczce Ardalis.SmartEnum, a wsparcie dla bibliotek stanowią osobne paczki np. Ardalis.SmartEnum.EFCore, Ardalis.SmartEnum.Dapper itd.
Pełna dokumentacja znajduje się pod linkiem: https://github.com/ardalis/SmartEnum.
Idea typu wyliczeniowego biblioteki jest identyczna z przedstawioną klasą w poprzedniej części. Należy zdefiniować własny typ dziedziczący po typie bazowym:

```
public sealed class ArdalisPlanet : SmartEnum<ArdalisPlanet>
{
   public static readonly double G = 6.67300E-11;

   public static readonly ArdalisPlanet Mars = new
ArdalisPlanet(nameof(Mars), 1, 6.421e+23, 3.3972e6);
   public static readonly ArdalisPlanet Mercury = new
ArdalisPlanet(nameof(Mercury), 2, 3.303e+23, 2.4397e6);
   public static readonly ArdalisPlanet Earth = new(nameof(Earth), 3,
5.976e+24, 6.37814e6);

public double Mass { get; private
set; }
 public double Radius { get;
private set; }

 private ArdalisPlanet(string name,
int value, double mass, double
radius): base(name, value)
 {
 Mass = mass;
 Radius = radius;
 }

 public double SurfaceGravity() {
 return G * Mass / (Radius *
Radius);
 }

 public double SurfaceWeight(double
otherMass) {
 return otherMass *
SurfaceGravity();
 }
}
```

W typie bazowym jest dodatkowa właściwość Value, która domyślnie jest typu int i może odpowiadać liczbie porządkowej stałej. W SmartEnum należy samodzielnie przypisać każdej stałej wartość Value. Możliwa jest też zmiana typu na inny, gdyż typ bazowy posiada dwa parametry generyczne.

```
public sealed class ArdalisPlanet :
SmartEnum<ArdalisPlanet, ushort>
{
 //deklaracje stałych
 //deklaracje właściwości
 private ArdalisPlanet(string name,
ushort value, double mass, double
radius): base(name, value)
 {
 Mass = mass;
 Radius = radius;
 }

 //deklaracje metod
}
```

SmartEnum posiadają podstawowe właściwości i metody statyczne charakterystyczne dla typów wyliczeniowych:

- List - zwraca kolekcję (tylko do odczytu) wszystkich stałych typu
- FromName, FromValue – metody zwracające stałą na podstawie odpowiednio nazwy i wartości.
- TryFromName, TryFromValue – metody zwracające wartość logiczną true jeśli stała o podanej nazwie lub wartości istnieje i zwraca stałą jako parametr wyjściowy metody.

Stałe wyliczeniowe posiadają metodę instancyjną When, która może zastąpić instrukcję switch.

```
planet
 .When(ArdalisPlanet.Mars)
 .Then(() => Console.WriteLine("Mars - the Red Planet"))
 .When(ArdalisPlanet.Earth)
 .Then(() => Console.WriteLine("Earth - the Green planet"))
 .When(ArdalisPlanet.Mercury)
 .Then(() => Console.WriteLine("Mercury - the Swift Planet"))
 .Default(() => Console.WriteLine("Other planet of the solar system"));
```

Pozostałe możliwości typu SmartEnum można poznać z dokumentacji.

Zakończenie

Typ wyliczeniowy w języku C# jest dość ubogi w stosunku do jego odpowiedników innych języków obiektowych. W prostych przypadkach można posiłkować się klasami rozszerzającymi i bazować na natywnym typie enum. W zaawansowanych przypadkach warto skorzystać z typu SmartEnum firmy Ardalis, który zawiera wsparcie dla wielu bibliotek i pozwala na tworzenie czystego, obiektowego kodu.