8- Návrhové vzory I

Základní

Jednoduchá tovární metoda (Simple factory method)

- Časté využití u neměnných immutable objektů
- Implementace pomocí statické metody
- Nahrazuje konsktruktor, když potřebujeme např.:
  - Rozhodnout, zda se má instance opravdu vytvořit
  - Vracet instance různých typů
  - Provést nějakou akci před zavoláním konstrukce
  - Více implementací se stejnými parametry (tovární metody se mohou jmenovat různě)
- Metoda se používá ve třídě, ze které dědí objekty, které metoda vytváří

```
public static Osoba GetInstance(int age, string name)
{
    if (age < 0) return null;
    if (age <= 7) return new Predskolak(age, name);
    if (age <= 19) return new Skolak(age, name);
    if (age <= 65) return new Pracujici(age, name);
    return new Duchodce(age, name);
}
```

Neměnný objekt (Immutable object)

- Objekty, který po vytvoření nejdou dále měnit (klíčové slovo readonly)

```
class MyClass
{
    private readonly string myStr;

    public MyClass(string str)
    {
        myStr = str;
    }

    public string GetStr
    {
        get { return myStr; }
    }
}
```

Přepravka (Crate, Transport Object)

- Předávání několika samostatných informací

- Atributy konstantní, aby byla zajištěna neměnnost

```
class Coordinate
{
    public readonly int x;
    public readonly int y;
    public readonly int z;

    public Coordinate(int x, int y, int z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}
```

Služebník (Servant)

- Přidání třídám další funkčnost bez úpravy rozhraní těchto tříd
- Tyto třídy nemusí mít společného rodiče, kam bychom definici funkcionality umístili
- Implementují ale stejné rozhraní, které je vstupním parametrem v metodách služebníka

Prázdný objekt (Null Object)

- Použití, když se nehodí klasická hodnota null
- Null Object je plnohodnotný objekt, ale prázdný; zastupuje hodnotu null
- Odpadá testování hodnot, zdali jsou null

```
abstract class Animal : IAnimal
{
    public abstract void Sound();

    public static readonly IAnimal Null = new NullAnimal();
    private class NullAnimal : Animal
    {
        public override void Sound() { }
    }
}

class Dog : IAnimal
{
    public void Sound()
    {
        WriteLine("Woof");
    }
}

// Main:
IAnimal dog = new Dog();
IAnimal fox = Animal.Null;
```

Pro vytváření instancí

Jedináček (Singleton)

- Může existovat pouze jedna instance dané třídy, která by měla být dostupná z více míst
- Existuje několik druhů implementace
- Instanci si lze vyžádat pomocí tovární metody (pokud instance neexistuje, je tovární metodou vytvořena)
- Samostatně se moc již nepoužívá, někteří jej považují za antipattern; je ale součástí jiných komplexnějších NV

```
class President
{
    private static President instance = null;
    private President() {}
    private static object lockThis = new object();

    public static President Instance
    {
        get
        {
            lock (lockThis)
            {
                if (instance == null)
                {
                    instance = new President();
                }
                return instance;
            }
        }
    }
}
```

Originál (Original)

- Instance jsou svými parametry unikátní
- Zajištění unikátnosti (nejsou dvě instance se stejnými parametry)

```
class Person
{
    private static List<Person> instances = new List<Person>();

    private string ID;

    private Person(string id)
    {
        ID = id;
    }

    public static Person Instance(string id)
    {
        Person instance = instances.Single(i => i.ID == id);
        if (instance == null)
        {
            instance = new Person(id);
            instances.Add(instance);
        }
        return instance;
    }
}
```

Fond (Pool)

- Umožnění znovupoužití existující instance třídy
- Řešení paměťově náročných tříd nebo tříd s velkým množstvím instancí
- Implementace pomocí kolekce, která bude obsahovat všechny instance
- Použití např. v bullet hell hrách

```
class Shelter
{
    private static int MAX_INSTANCES = 0;
    private static Cat[] instances = new Cat[MAX_INSTANCES];

    public Shelter(int maxInstances)
    {
        MAX_INSTANCES = maxInstances;
    }

    public Cat GetInstance(string[] params)
    {
        if (instances.length < MAX_INSTANCES)
        {
            Cat instance = new Cat(params);
            instances.Append(instance);
            return instance;
        }
        return null;
    }
}
```

Muší váha (Flyweight)

- Minimalizuje využití paměti tím, že mezi podobnými objekty sdílí co nejvíce informací
- Využití u tříd s velkým množstvím instancí
- Společná část musí být immutable
- Realtime strategy games