9- Návrhové vzory I

Skrývání implementace

Zástupce (Proxy)

- Nahradí objekt jiným zástupným objektem
- Vizuálně stejný, ale můžeme zasahovat do jeho chování
- Umožní zachytávat a převádět volání na jiná
- Implementován rozhraním nebo abstraktní třídou
- Vzdálený
    - Zastupuje objekt, který se nachází jinde
    - Zprostředkovává komunikaci mezi zastupovaným vzdáleným objektem a objekty z okolí zástupce
    - Musí být stále připraven na možnost selhání
    - Bankomat komunikující s bankou
- Ochranný
    - Kontrola přístupu k datům na základě práv
- Virtual
    - Používá se, když potřebujeme odložit vytváření objektu (velký obrázek je nahrazen virtual proxy, zatímco se načítá)

Příkaz (Command)

- Zabalí metodu do objektu
- Reakce na určitou akci
- Oddělení spouštěcí akce od akce činící

```
<Button Command="{Binding StartListening}"/>


2 references
public RelayCommand StartListening { get; set; }

1 reference
public ListenerViewModel()
{
    StartListening = new RelayCommand(
        () =>
        {
            Listen();
        }
        );
}
```

Iterátor (Iterator)

- Samostatný objekt umožňující lineární procházení kolekcemi bez znalosti jejich vnitřní implementace
- V C# na kolekce implementující IEnumerator lze použít foreach = **Implicitní iterátor**
- **Explicitní iterátor** si musíme sami napsat, instance iterátoru na požádání vrací odkaz na další prvek v kolekci, dokud nedojede k prvku poslednímu
- **Externí iterátor**: průchod kolekcí řídí klient s využitím iterátoru; složitější, ale mocnější řešení
- **Interní iterátor**: průchod kolekcí řídí iterátor sám; klient specifikuje, co provádět s prvky kolekce (např pomocí LINQ + lambda)

```
1 reference
public interface IMyIterator<T>
{
    1 reference
    bool HasNext();
    1 reference
    T Next();
}
1 reference
public class ArrayIterator<T> : IMyIterator<T>
{
    private T[] _array;
    private int index;
    0 references
    public ArrayIterator(T[] array)
    {
        _array = array;
        index = -1;
    }
    1 reference
    public bool HasNext()
    {
        return index < _array.Length - 1 ? true : false;
    }
    1 reference
    public T Next()
    {
        if(index > 0)
        {
            index++;
        }
        return _array[index];
    }
}
```

```
// ZKRATKOU //
using _9_NVII.Iterator;
string[] Zoo = { "Slon", "Zirafa", "Mamut", "Zebra", "Piskomil" };

ArrayIterator<string> iterator = new ArrayIterator<string>(Zoo);

while (iterator.HasNext())
{
    Console.WriteLine(iterator.Next());
}
```

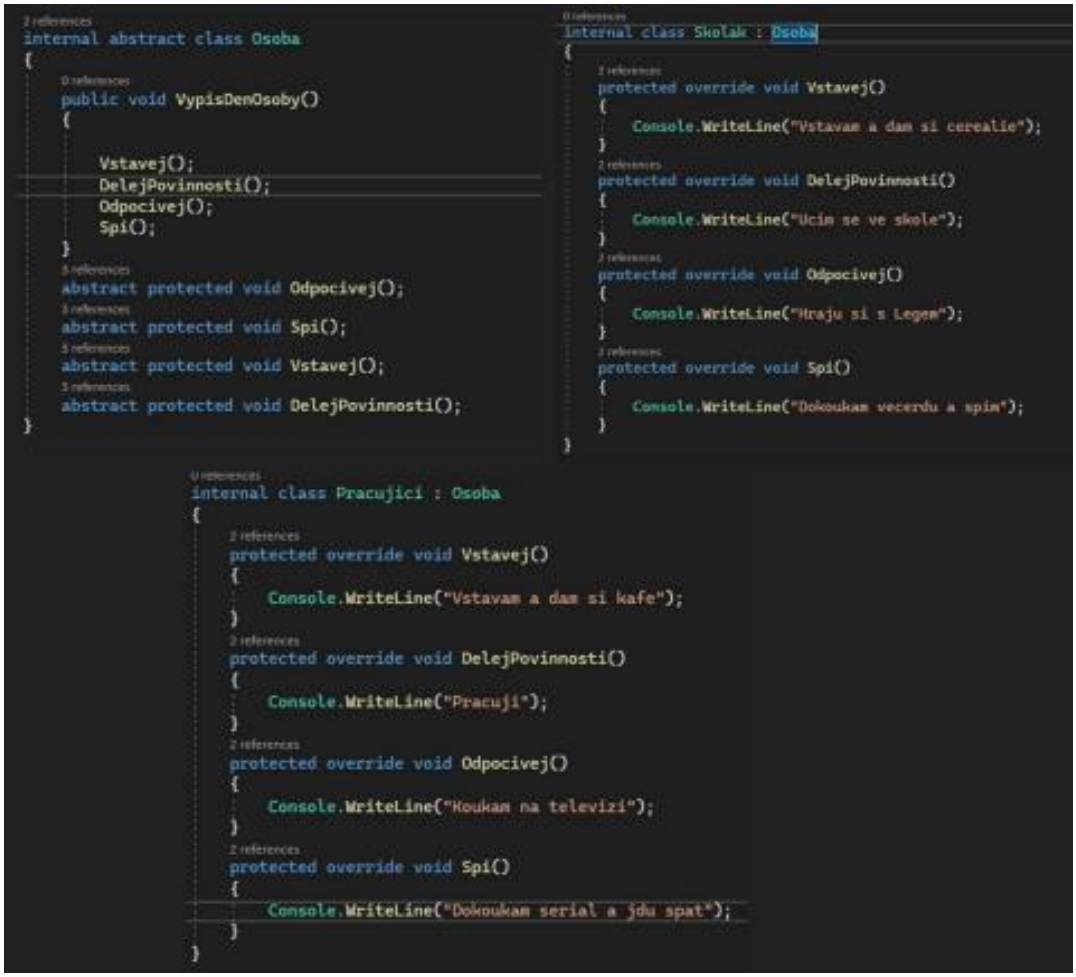
Stav (State)

- Řeší velké rozdíly v chování objektu v určitých stavech
- Objekt má konečný počet definovaných stavů a vždy se může nacházet v jednom z nich
- Animace v Unity

Šablonová metoda (Template Method)

- Definuje „kostru“ algoritmu

- Neobsahuje všechny kroky – prázdné virtuální metody pro přepsání potomky
- Mění části algoritmu beze změny původní části algoritmu
- Způsob odstranění duplicit v kódu
- Pro úlohy s parametry, které budou známy až za běhu
- Použití:
    - Když chceme, aby se naše aplikace dala rozšířit, ale ne modifikovat
    - Když se nám v kódu objevují podobné algoritmy (lišící se jen v pár krocích, kostra je stejná)



```
internal abstract class Osoba
{
    public void VypisDenOsoby()
    {
        Vstavej();
        DelejPovinnosti();
        Odpocivej();
        Spi();
    }
    abstract protected void Odpocivej();
    abstract protected void Spi();
    abstract protected void Vstavej();
    abstract protected void DelejPovinnosti();
}
```

```
internal class Skolak : Osoba
{
    protected override void Vstavej()
    {
        Console.WriteLine("Vstavam a dam si cerealie");
    }
    protected override void DelejPovinnosti()
    {
        Console.WriteLine("Ucim se ve skole");
    }
    protected override void Odpocivej()
    {
        Console.WriteLine("Hraju si s Legem");
    }
    protected override void Spi()
    {
        Console.WriteLine("Dokoukam vecerdu a spim");
    }
}
```

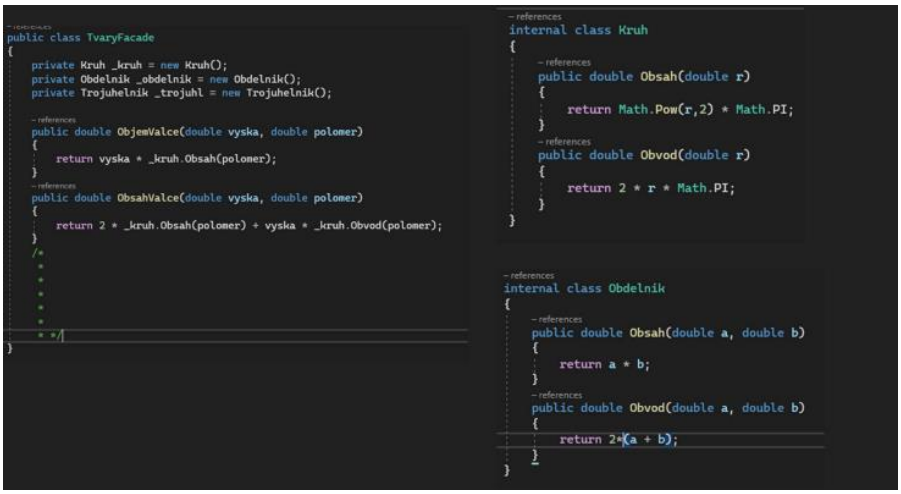
```
internal class Pracujici : Osoba
{
    protected override void Vstavej()
    {
        Console.WriteLine("Vstavam a dam si kafe");
    }
    protected override void DelejPovinnosti()
    {
        Console.WriteLine("Pracuji");
    }
    protected override void Odpocivej()
    {
        Console.WriteLine("Koukam na televizi");
    }
    protected override void Spi()
    {
        Console.WriteLine("Dokoukam serial a jdu spat");
    }
}
```

Optimalizace rozhraní

Fasáda (Facade)

- Zjednodušuje komunikaci mezi uživatelem a systémem
- Vytvoření jednotného rozhraní pro celou logickou skupinu tříd, které se tak sdruží do subsystému, který je většinou příliš složitý (toolbox, imagine brát si nářadí s jedné hromady)

- Zabalí komplikovaný subsystém do jednoduššího uceleného rozhraní



```
public class TvaryFacade
{
    private Kruh _kruh = new Kruh();
    private Obdelnik _obdelnik = new Obdelnik();
    private Trojuhelnik _trojuhl = new Trojuhelnik();

    public double ObjemValce(double vyska, double polomer)
    {
        return vyska * _kruh.Obsah(polomer);
    }
    public double ObsahValce(double vyska, double polomer)
    {
        return 2 * _kruh.Obsah(polomer) + vyska * _kruh.Obvod(polomer);
    }
    /*
     *
     *
     *
     *
     *
     * */
}
```

```
internal class Kruh
{
    public double Obsah(double r)
    {
        return Math.Pow(r,2) * Math.PI;
    }
    public double Obvod(double r)
    {
        return 2 * r * Math.PI;
    }
}
```

```
internal class Obdelnik
{
    public double Obsah(double a, double b)
    {
        return a * b;
    }
    public double Obvod(double a, double b)
    {
        return 2*(a + b);
    }
}
```

Adaptér (Adapter)

- Převede zastaralé / nehodící se / chybné rozhraní třídy na rozhraní, které klient očekává
- Zabezpečuje spolupráci tříd a usnadňuje implementaci nových
- Může celou třídu zabalit do nové (object adaper), nebo z ní dědit (class adapter)

Strom (Composite)

- Popisuje skupinu objektů, se kterými je zacházeno stejně, jako s jednou instancí tohoto objektu
- => ignorování rozdílu mezi vnořenými objekty a samostatným objektem
- Složka která obsahuje další prvky, což mohou být další složky