DESIGN PATTERNS INTRODUCTIE

WEEK 1

Interface Opdracht: “NumberConverter”

Deze week ga je een interface maken en deze toepassen op een aantal concrete klassen. Vervolgens ga je met die interface werken in een aanroepende klasse. We gebruiken hier het **Strategy pattern.**

Daarnaast ga je er voor zorgen dat het viewmodel de verschillende strategies niet kent. Dit gaan we doen door middel van het **Factory Method pattern**.

1. Clone de repository en run de code.

Clone het project https://github.com/Avans/DPINT_Wk1_Strategies.
Hier zie je een window met 2 velden, een button en 2 dropdown velden.
Kies links voor een nummerformaat, vul een nummer in en kijk wat er rechts gebeurt als je op de knop drukt. We kunnen de nummers transformeren van het ene naar het andere formaat.

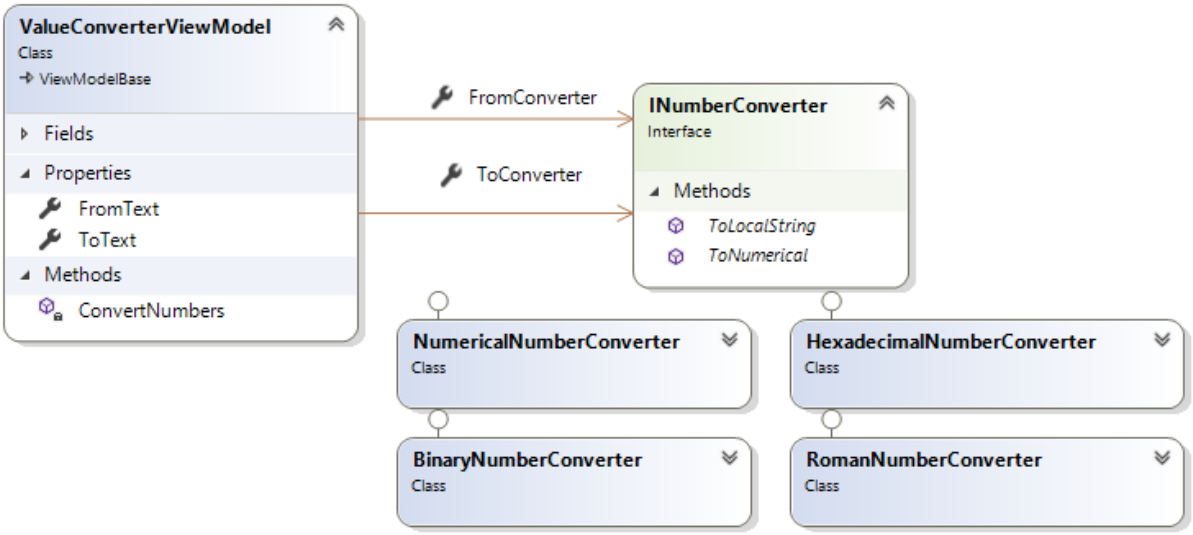
In **ValueConverterViewModel** zit alle logica. Deze gaan we dan ook aanpassen.

2. Creëer de klassen en interfaces uit onderstaand diagram

We willen graag het strategy pattern implementeren. Hiervoor hebben we de interface INumberConverter nodig. Deze kent twee methodes: ToLocalString en ToNumerical.

De ValueConverterViewModel kent dan 2 instanties van die interface, _fromConverter en _toConverter.

Pas de code aan (trek de code uit de methode ConvertNumbers() uit elkaar in verschillende klasses) zodat deze matcht met onderstaand model.



WEEK 1

3. Pas de ConvertNumbers() methode aan

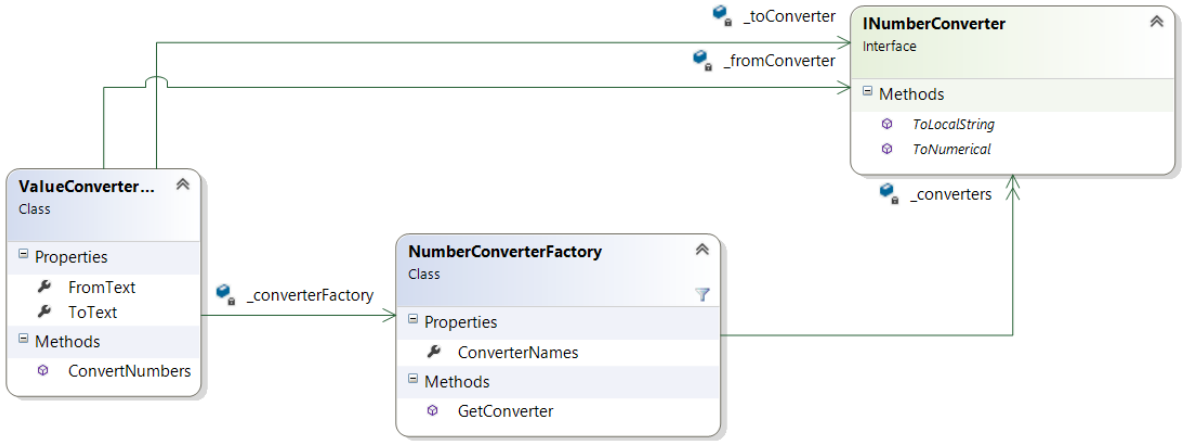
De methode ConvertNumbers() kent nu mooi geen switches meer!
Let op: Je applicatie werkt nu niet meer.

```
public void ConvertNumbers()
{
    int number = _fromConverter.ToNumerical(FromText);
    ToText = _toConverter.ToLocalString(number);
}
```

4. Hoe komen we nu aan de juiste converters?

We zijn nu nog steeds afhankelijk van strings in de ValueConverterViewModel . Dit is niet wenselijk, als we een nieuwe converter hebben willen we deze klasse niet aan hoeven te passen!
We gaan daarom hier een **Factory Method** inbouwen.

De factory method gaat het viewmodel voorzien van de namen van de converters en van de converters zelf! Hij is de enige die de implementerende klasses van de interface INumberConverter kent.
Let op: Je applicatie werkt nog steeds niet.



```
class NumberConverterFactory
{
    public IEnumerable<string> ConverterNames
    {
        get { return _converters.Keys; }
    }

    private Dictionary<string, INumberConverter> _converters;

    public NumberConverterFactory()
    {
        _converters = new Dictionary<string, INumberConverter>();

        _converters["Numerical"] = new NumericalNumberConverter();
        // ...en de andere converters nog toevoegen.
    }

    public INumberConverter GetConverter(string name)
    {
        // Hoe geven we nu de juiste converter terug?
        throw new NotImplementedException();
    }
}
```

5. Pas het viewmodel aan

Het viewmodel moet in de constructor aangepast worden. Hij moet een nieuwe factory aanmaken en de beschikbare namen van de factory in de ConverterNames stoppen.

Als de dropdown aangepast wordt, moet natuurlijk óf de _fromConverter óf de _toConverter vervangen worden. Deze kunnen we aan de factory vragen!
Tip: Kijk in de setters van FromConverterName en ToConverterName. Daar kan je de _fromConverter en de _toConverter setten door deze aan de factory op te vragen.

Als het goed is werkt alles nu en is het viewmodel **volledig** onafhankelijk van de implementerende logica!

6. Voeg een nieuwe converter toe

Maak nu de klasse OctalNumberConverter, deze gebruikt de volgende methode:

```
Convert.ToInt32(FromText, 8);
```

Voeg deze toe aan je factory en zie dat ook deze beschikbaar is zonder het viewmodel aangepast te hebben!