Transformation Priority Premise

Sindre Smistad

Transformation Priority Premise

- Not a refactoring
- Simple operations that change behaviour
- Can be used to pass red/green/refactor cycle
- Have a priority
- We want to avoid rewriting whole method to pass a new test
- Look for test cases that can be solved with a higher priority transformation

Transformation Priority Premise - What is “Obvious implementation” ?

#	TRANSFORMATION	STARTING CODE	FINAL CODE
1	{} => **nil**		`return nil`
2	nil => **constant**	`return nil`	`return “1”`
3	constant => **constant+**	`return “1”`	`return “1” + “2”`
4	constant => **scalar**	`return “1” + “2”`	`return argument`
5	statement => **statements**	`return argument`	`return arguments`
6	unconditional => **conditional**	`return arguments`	`if(condition)return arguments`
7	scalar => **array**	`dog`	`[dog, cat]`
8	array => **container**	`[dog, cat]`	`{dog = “DOG”, cat = “CAT”}`
9	statement => **recursion**	`a + b`	`a + recursion`
10	conditional => **loop**	`if(condition)`	`while(condition)`
11	recursion => **tail recursion**	`a + recursion`	`recursion`
12	expression => **function**	`today - birthday`	`CalculateAge()`
13	variable => **mutation**	`day`	`var day = 10; day = 11;`
14	**switch case**		

Roman Numeral Converter - TPP

```
// Test - Convert(1)
public string Convert(int arabicNumber)
{
    return "I";  // Stupid simple
}
```

Roman Numeral Converter - TPP

```
// [TestCase(1, “I”)]
// [TestCase(2, “II”)]
// [TestCase(3, “III”)]
public string Convert(int arabicNumber)
{
    if (arabicNumber == 1)
        return "I";
    if (arabicNumber == 2)
        return "I" + "I";
    return "I" + "I" + "I";     // Constant +
}
```

Roman Numeral Converter - TPP

```
// [TestCase(1, “I”)]
// [TestCase(2, “II”)]
// [TestCase(3, “III”)]
public string Convert(int arabicNumber)
{
    if (arabicNumber == 1)
        return "I";
    if (arabicNumber == 2)
        return "I" + "I";
    return "I" + "I" + "I";      // Constant +
}                                // Time to triangulate
```

Roman Numeral Converter - TPP

```
 var lookUp = new string[] { "I", "II", "III" }; // Array is priority 7, looping is 11
return lookUp[arabicNumber - 1];
```

Roman Numeral Converter - TPP

```
if (arabicNumber == 6)
{
    return "V" + "I";
}
```

```
    var arabicToRoman = new Dictionary<int, string>    // container
            { { 1, "I" }, { 4, "IV" }, {5, "V"} };

if (arabicNumber == 6)
{
    return arabicToRoman[5] + arabicToRoman[1];    // scalar
}
```

Roman Numeral Converter - TPP

```
if (arabicNumber == 6)                          // Hmmm... Things seems to be repating
{
    return arabicToRoman[5] + arabicToRoman[1];
}
if (arabicNumber == 7)
{
    return arabicToRoman[5] + arabicToRoman[1] + arabicToRoman[1];
}
if (arabicNumber == 8)
{
    return arabicToRoman[5] + arabicToRoman[1] + arabicToRoman[1] + arabicToRoman[1];
}
```

Roman Numeral Converter - TPP

```
var result = string.Empty;
if (arabicNumber > 5) // Numbers above 5 is really “V” + “I” * remainder
{
    result = arabicToRoman[5];
    for (var i = 0; i < arabicNumber-5; i++)
        result += arabicToRoman[1];

    return result;
}
for (var i = 0; i < arabicNumber; i++)
    result += arabicToRoman[1];
```

Roman Numeral Converter - TPP

```
var result = string.Empty;
int remainder = arabicNumber;           // We made remainder a constant, removed loop
if (arabicNumber > 5)
{
    result = arabicToRoman[5];
    remainder = arabicNumber-5;

}
for (var i = 0; i < remainder; i++)
    result += arabicToRoman[1];
```

Roman Numeral Converter - TPP

```
.
.
.
while (remainder >= 1)
{
    result += arabicToRoman[1];
    remainder = remainder - 1;
}
```

```
// Finding the concept in the end
```

Roman Numeral Converter - TPP

```
var result = string.Empty;
int remainder = arabicNumber;                                   // Destination
foreach (var number in _arabicToRoman.Keys)
    while (remainder >= number)
    {
        result += _arabicToRoman[number];
        remainder -= number;
    }
return result;
```