

THE TRANSFORMATION PRIORITY PREMISE

TPP

WHAT IS TPP

- ▶ A programming approach introduced by Robert C. Martin (Uncle Bob) supports the TDD process suggests that refactoring have a counterpart "Transformations" transformations are simple Operations that change the behaviour of code makes TDD more effective for a computer programmer
- ▶ In the red/green/refactor cycle every change to the code is either a behavior changing transformation from specific to generic or a refactoring.
- ▶ The TPP transformations at the top of the list are simpler and less risky, than the transformations that are lower in the list
- ▶ let's dig into the transformations of the transformation priority premise

TPP RULES (SHORT EXPLANATION WITH A BIT OF CODE)

- ▶ (`{}`->null) no code at all => return null

```
public String convert(int number) {  
    return null;  
}
```

- ▶ (null->constant) return a constant instead "null"

```
public String convert(int number) {  
    return "I";  
}
```

- ▶ (constant->constant+) simple constant to more complex constant

```
public String convert(int number) {  
    return "I" + "I";  
}
```

Note: This transformation is not sufficient to make the test (convert_2_to_II) pass

- ▶ (constant->scalar) replace constant with variable or argument

```
public String convert(int number) {  
    String result = "I";  
    return result;  
}
```

Note: This transformation is not sufficient to make the test (convert_2_to_II) pass

TPP RULES (SHORT EXPLANATION WITH A BIT OF CODE)

▶ (statement->statements) adding more unconditional statements

```
public String convert(int number) {  
    String result = "I";  
    result += "I";  
    return result;  
}
```

Note: This transformation is not sufficient to make the test (convert_2_to_II) pass

▶ (unconditional->if) splitting the execution path

```
public String convert(int number) {  
    String result = "I";  
    if(number > 1) result += "I";  
    return result;  
}
```

Note: This transformation is sufficient to make the test (convert_2_to_II) pass

▶ (variable->array)

```
public static String[] results = {"I", "II", "III"};  
public String convert(int number) {  
    return results[number - 1];  
}
```

▶ Note: Based on the "Rule of three"

TPP RULES (SHORT EXPLANATION WITH A BIT OF CODE)

▶ (array->container)

```
public static Map<Integer,String> results = new HashMap<Integer, String>(){  
    put(1,"I");  
    put(2, "II");  
    put(3, "III");  
    put(4, "IV");  
};  
public String convert(int number) {  
    return results.get(number);  
}
```

▶ (statement ->recursion)

```
public static Map<Integer,String> results = new HashMap<Integer, String>(){  
    put(1,"I");  
    put(4, "IV");  
};
```

```
public String convert(int number) {  
    if(results.containsKey(number)) {  
        return results.get(number);  
    }  
    return results.get(1) + convert(number - 1);  
}
```



TPP RULES (SHORT EXPLANATION WITH A BIT OF CODE)

▶ (if->while)

```
public String convert(int number) {
```

```
    int numberToConvert = number;
    if(results.containsKey(number)) {
        return results.get(number);
    }
    String result = "";
```

```
    while(numberToConvert >= 40) {
        result += results.get(40);
        numberToConvert -= 40;
    }
    ...
```

▶ (Statement->recursion)

```
public String convert(int number) {
```

```
    if(results.containsKey(number)) {
        return results.get(number);
    }
    if(number > 40) {
        String result = results.get(40);
        return result + convert(number - 40);
    }
    ...
```

Rule of three - repetition of the while loop - refactoring

TPP RULES (SHORT EXPLANATION WITH A BIT OF CODE)

▶ (if->while)

```
public String convert(int number) {
```

```
    int numberToConvert = number;
    if(results.containsKey(number)) {
        return results.get(number);
    }
    String result = "";
```

```
    while(numberToConvert >= 40) {
        result += results.get(40);
        numberToConvert -= 40;
    }
    ...
}
```

Rule of three -
repetition of the while loop - refactoring

▶ (Statement->recursion)

```
public String convert(int number) {
```

```
    if(results.containsKey(number)) {
        return results.get(number);
    }
    if(number > 40) {
        String result = results.get(40);
        return result + convert(number - 40);
    }
    ...
}
```

```
public String convert(int number) {
    int numberToConvert = number;
    String result = "";
    final Iterator<Map.Entry<Integer, String>> mappingIterator = ... iterator();
    while(mappingIterator.hasNext()) {
        Map.Entry<Integer, String> current = mappingIterator.next();
        while(numberToConvert >= current.getKey()) {
            result += current.getValue();
            numberToConvert -= current.getKey();
        }
    }
    return result;
}
```

TPP RULES (SHORT EXPLANATION WITH A BIT OF CODE)

- ▶ (expression -> function)

No code found in by example - but maybe it could be used for the concat operations

- ▶ (variable -> assignment)

- ▶ Uncle Bob explicitly stated -> There are likely others