Kata:
Convert roman to arabic numbers using Transformation Priority Premise

The Transformations (1)

- [] –> nil
- nil –> constant
- constant –> constant+
- constant –> scalar
- statement –> statements
- unconditional –> if

The Transformations (2)

- scalar –> array
- array –> container
- statements –> tail recursion
- if –> loop
- statement → recursion
- expression –> function
- variable –> assignment

Kata

- Write a programm to convert roman numbers to arabic

Roman	Arabic	Roman	Arabic
I	1	XX	
II	2	DCCCXLVI	846
III	3		
IV	4		
V	5		
IX	9		
XII	11		

{} → nil

```
// test first

class ArabicConverterShouldReturn {

    @ParameterizedTest
    @CsvSource({
            "1, I"
    })
    void ArabicNumberForRomanString(Integer arabic, String roman) {
        ArabicConverter arabicConverter = new ArabicConverter();

        Integer returnValue = arabicConverter.convertToArabic(roman);

        assertEquals(arabic, returnValue);
    }
}

// 1. {} → nil

public class ArabicConverter {
    public Integer convertToArabic(String i) {
        return null;
    }
}

// test fails
```

nil → constant

```
// 2. nil → constant

public class ArabicConverter {
    public Integer convertToArabic(String i) {
        return 1;
    }
}

// test passes

// new failing test

    @ParameterizedTest
    @CsvSource({
            "1, I",
            "2, II",
    })

// 3. (constant → constant+) // not sufficient

// 4. (constant → variable) // not sufficient

public class ArabicConverter {
    public Integer convertToArabic(String roman) {
        Integer arabic = 1;
        return arabic;
    }
}

// test still failing
```

unconditional → conditional

```
// 5. statement -> statements // not sufficient

public Integer convertToArabic(String roman) {
    Integer arabic = 1;

    arabic += 1;
    return arabic;
}

// test still failing

// 6. unconditional -> conditional

public class ArabicConverter {
    public Integer convertToArabic(String roman) {
        Integer arabic = 1;
        if (roman.equals("II")) {
            arabic += 1;
        }
        return arabic;
    }
}

// test passes
```

code duplication

```
// add a new failing test

@ParameterizedTest
@CsvSource({
        "1, I",
        "2, II",
        "3, III"
})

// still on step 6 unconditional -> conditional

public class ArabicConverter {
    public Integer convertToArabic(String roman) {
        Integer arabic = 1;
        if (roman.equals("II")) {
            arabic = 2;
        }
        if (roman.equals("III")) {
            arabic = 3;
        }
        return arabic;
    }
}

// test passes but we have code duplication
```

works for the moment

```
public class ArabicConverter {
    public Integer convertToArabic(String roman) {

         return roman.length();
    }
}

// test passes

// add new failing test

@CsvSource({
        "1, I",
        "2, II",
        "3, III",
        "4, IV"
})

// adapt the code

public class ArabicConverter {
    public Integer convertToArabic(String roman) {

        if (roman.equals("IV")) {
            return 4;
        }
        return roman.length();
    }
}
```

one step after the other

```
// add new failing test

@ParameterizedTest
@CsvSource({
        "1, I",
        "2, II",
        "3, III",
        "4, IV",
        "5, V"
})

// adapt the code

public Integer convertToArabic(String roman) {

    if (roman.equals("IV")) {
        return 4;
    }
    if (roman.equals("V")) {
        return 5;
    }
    return roman.length();
}

// test passes but code duplication

// 7 variable -> array (not sufficient)

// 8 array -> container
```

a map for the rescue

```
public class ArabicConverter {
    static Map<String, Integer> arabics = new HashMap<>();

    static {
        arabics.put("I", 1);
        arabics.put("II", 2);
        arabics.put("III", 3);
        arabics.put("IV", 4);
        arabics.put("V", 5);
    }

    public Integer convertToArabic(String roman) {

        return arabics.get(roman);
    }
}

// test passes

// new failing test

"6, VI"

// add to map

arabics.put("VI", 6);
```

statement → tail recursion

```
      // test passes but code duplication

      // 9. statement → tail recursion (First Approach)

public class ArabicConverter {
    static Map<Character, Integer> arabics = new HashMap<>();

    static {
        arabics.put('I', 1);
        arabics.put('V', 5);
    }

    public Integer convertToArabic(String roman, Integer sum) {

        Integer currentValue = arabics.get(roman.charAt(0));
        if (roman.length() == 1) {
            return sum + currentValue;
        }

        return  convertToArabic(roman.substring(1), sum);
    }
}

// test for IV failing: 1 + 5 <> 4 (adding values not enough)
```

Last Obstacle

// if the value of a roman letter right from the current one is bigger we have to subtract (next_val – cur_val) e.g.

IV → 5 – 1

// or multiply with (-1)

(-1) * 1 + 5

Final approach

```
public class ArabicConverter {
    static Map<Character, Integer> arabics = new HashMap<>();

    static {
        arabics.put('I', 1);
        arabics.put('V', 5);
        arabics.put('X', 10);
        arabics.put('L', 50);
        arabics.put('C', 100);
        arabics.put('D', 500);
        arabics.put('M', 1000);
    }

    public Integer convertToArabic(String roman, Integer sum) {

        Integer currentValue = arabics.get(roman.charAt(0));
        if (roman.length() == 1) {
            return sum + currentValue;
        }

        sum += invertValueIfSmallerThanNextOne(currentValue, arabics.get(roman.charAt(1)));
        return  convertToArabic(roman.substring(1), sum);
    }

    private Integer invertValueIfSmallerThanNextOne(Integer current, Integer next) {
        if (current < next) {
            return current * -1;
        }
        return current;
    }
}
```

Conclusion

TPP gives a helpful guideline to find the next steps during refactoring