Roman Numbers Kata

Roman Numbers.

Given a positive integer number (eg. 42) determine its Roman numeral representation as a String (eg "XLII"). You cannot write numerals like IM for 999.

Arabic number	Roman numeral	Arabic number	Roman numeral
1	I	60	LX
2	II	70	LXX
3	III	80	LXXX
4	IV	90	XC
5	V	100	C
6	VI	200	CC
7	VII	300	CCC
8	VIII	400	CD
9	IX	500	D
10	X	600	DC
20	XX	700	DCC
30	XXX	800	DCCC
40	XL	900	CM
50	L	1000	M

Arabic number	Roman numeral	Thousands	Cents	Thenths	Units
846	DCCCXLVI	-	DCC		
1999	MCMXCIX	M	CM		
2008	MMVIII				

The Requirements.

- Write a class named “RomanNumbers” that has one method
  - convert(number: int) is called to convert an Arabic number to a Romanian number.

Begin.

- Create a project named RomanNumbers
- Create a unit test named RomanConverterShould

```
public class RomanConverterShould {
}
```

The first test.

```java
import org.junit.Test;

public class RomanConverterShould {

    @Test
    public void converNumberToRoman() {
        RomanConverter romanConverter = new RomanConverter();
    }
}
```

The first test.

```java
import org.junit.Test;

public class RomanConverterShould {

    @Test
    public void converNumberToRoman() {
        RomanConverter romanConverter = new RomanConverter();
    }
}
```

```java
public class RomanConverter {

}
```

The first test.

```java
import junitparams.JUnitParamsRunner;
import junitparams.Parameters;
import org.junit.Test;
import org.junit.runner.RunWith;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.CoreMatchers.is;

@RunWith(JUnitParamsRunner.class)
public class RomanConverterShould {

    @Test
    @Parameters({"1, I"})
    public void converNumberToRoman(int number, String expected) {
        RomanConverter romanConverter = new RomanConverter();
        assertThat(romanConverter.convert(number), is(expected));
    }
}
```

```java
public class RomanConverter {

}
```

The first test.

```java
import junitparams.JUnitParamsRunner;
import junitparams.Parameters;
import org.junit.Test;
import org.junit.runner.RunWith;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.CoreMatchers.is;

@RunWith(JUnitParamsRunner.class)
public class RomanConverterShould {

    @Test
    @Parameters({"1, I"})
    public void converNumberToRoman(int number, String expected) {
        RomanConverter romanConverter = new RomanConverter();
        assertThat(romanConverter.convert(number), is(expected));
    }
}
```

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

The first test.

```java
import junitparams.JUnitParamsRunner;
import junitparams.Parameters;
import org.junit.Test;
import org.junit.runner.RunWith;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.CoreMatchers.is;

@RunWith(JUnitParamsRunner.class)
public class RomanConverterShould {

    @Test
    @Parameters({"1, I"})
    public void converNumberToRoman(int number, String expected) {
        RomanConverter romanConverter = new RomanConverter();
        assertThat(romanConverter.convert(number), is(expected));
    }
}
```

```
expected:"I" but was null
```

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

The first test.

```java
import junitparams.JUnitParamsRunner;
import junitparams.Parameters;
import org.junit.Test;
import org.junit.runner.RunWith;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.CoreMatchers.is;

@RunWith(JUnitParamsRunner.class)
public class RomanConverterShould {

    @Test
    @Parameters({"1, I"})
    public void converNumberToRoman(int number, String expected) {
        RomanConverter romanConverter = new RomanConverter();
        assertThat(romanConverter.convert(number), is(expected));
    }
}
```

```java
public class RomanConverter {

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

```
Nil to Constant: pass
```

The second test.

```
…
@Test
@Parameters({
        "1, I",
        "2, II"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

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

The second test.

```
…
@Test
@Parameters({
        "1, I",
        "2, II"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String convert(int number) {

        String result = "I";

        return result;
    }
}
```

Constant to variable: fail

The second test.

```
…
@Test
@Parameters({
        "1, I",
        "2, II"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String convert(int number) {

        String result = "I";
        result += "I";

        return result;
    }
}
```

```
Statement to statements: fail
```

The second test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String convert(int number) {

        String result = "I";
        if (number > 1) {
            result += "I";
        }

        return result;
    }
}
```

Unconditional to conditional: pass

The third test.

```
…
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String convert(int number) {

        String result = "I";
        if (number > 1) {
            result += "I";
        }

        return result;
    }
}
```

The third test.

```
…
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String convert(int number) {

        String result = "I";
        if (number > 1) {
            result += "I";
        }

        return result;
    }
}
```

```
expected:”III” but was “I”
```

- If statement is duplicated

The third test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String convert(int number) {

        String result = "I";
        if (number > 1) {
            result += "I";
        }

        if (number > 2) {
            result += "I";
        }

        return result;
    }
}
```

The third test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String[] results = new String[] {"I", "II", "III"};

    public String convert(int number) {

        return results[number - 1];

    }
}
```

Variable to array: pass

The fourth test.

```
…
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String[] results = new String[] {"I", "II", "III"};

    public String convert(int number) {

        return results[number - 1];

    }
}
```

The fourth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String[] results = new String[] {"I", "II", "III"};

    public String convert(int number) {

        return results[number - 1];

    }
}
```

ArrayOutOfBoundException

- "I" is duplicated

The fourth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    public String[] results = new String[] {"I", "II", "III", "IV"};

    public String convert(int number) {

        return results[number - 1];

    }
}
```

- "I" is duplicated

The fourth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    private HashMap<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);

    }
}
```

Array to collection: pass

The fourth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    private HashMap<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);
    }
}
```

Statement to tail recursion: pass

- More duplicated "I"

The eighth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV",
        "5, V",
        "6, VI",
        "7, VII",
        "8, VIII"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    private HashMap<Integer, String> results = new HashMap<Integer, String>() {
        {
            put(1, "I");
            put(4, "IV");
            put(5, "V");
            put(6, "VI");
            put(7, "VII");
            put(8, "VIII"); // more duplicated I
        }
    };

    public String convert(int number) {

        if (results.containsKey(number)) {
            return results.get(number);
        }

        return results.get(1) + convert(number - 1);
    }
}
```

The eighth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV",
        "5, V",
        "6, VI",
        "7, VII",
        "8, VIII"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    private HashMap<Integer, String> results = new HashMap<Integer, String>() {
        {
            put(1, "I");
            put(4, "IV");
            put(5, "V");
        }
    };

    public String convert(int number) {

        if (results.containsKey(number)) {
            return results.get(number);
        }

        if (number > 5) {
            String result = "V";
            return result + convert(number - 5);
        }

        return results.get(1) + convert(number - 1);
    }
}
```

Statement to tail recursion: pass

The twelveth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV",
        "5, V",
        "6, VI",
        "7, VII",
        "8, VIII",
        "9, IX",
        "10, X",
        "40, XL",
        "44, XLIV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    private HashMap<Integer, String> results = new HashMap<Integer, String>() {
        {
            put(1, "I");
            put(4, "IV");
            put(5, "V");
            put(9, "IX");
            put(10, "X");
            put(40, "XL");
        }
    };

    public String convert(int number) {

        if (results.containsKey(number)) {
            return results.get(number);
        }

        if (number > 5) {
            String result = "V";
            return result + convert(number - 5);
        }

        return results.get(1) + convert(number - 1);
    }
}
```

```
expected:"XLIV" but was "VVVVVVVIX"
```

- If statement duplicated

The twelveth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV",
        "5, V",
        "6, VI",
        "7, VII",
        "8, VIII",
        "9, IX",
        "10, X",
        "40, XL",
        "44, XLIV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {
    private HashMap<Integer, String> results = new HashMap<Integer, String>() {
        {
            put(1, "I");
            put(4, "IV");
            put(5, "V");
            put(9, "IX");
            put(10, "X");
            put(40, "XL");
        }
    };

    public String convert(int number) {

        if (results.containsKey(number)) {
            return results.get(number);
        }

        if (number > 40) {
            String result = "XL";
            return result + convert(number - 40);
        }

        if (number > 10) {
            String result = "X";
            return result + convert(number - 10);
        }

        if (number > 5) {
            String result = "V";
            return result + convert(number - 5);
        }

        return results.get(1) + convert(number - 1);
    }
}
```

Statement to tail recursion: pass

- While is duplicated

The twelveth test.

```
…
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV",
        "5, V",
        "6, VI",
        "7, VII",
        "8, VIII",
        "9, IX",
        "10, X",
        "40, XL",
        "44, XLIV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

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

        if (results.containsKey(number))
        {
            return results.get(number);
        }

        String result = "";

        while (number >= 40)
        {
            result += "XL";
            number -= 40;
        }

        while (number >= 10)
        {
            result += "X";
            number -= 10;
        }

        while (number >= 5)
        {
            result += "V";
            number -= 5;
        }

        while (number >= 4)
        {
            result += "IV";
            number -= 4;
        }

        while (number >= 1)
        {
            result += "I";
            number -= 1;
        }

        return result;
    }
}
```

If to while: pass

The twelveth test.

```
...
@Test
@Parameters({
        "1, I",
        "2, II",
        "3, III",
        "4, IV",
        "5, V",
        "6, VI",
        "7, VII",
        "8, VIII",
        "9, IX",
        "10, X",
        "40, XL",
        "44, XLIV"
})
public void converNumberToRoman(int number, String expected) {
    RomanConverter romanConverter = new RomanConverter();
    assertThat(romanConverter.convert(number), is(expected));
}
```

```
public class RomanConverter {

    private Map<Integer, String> results = new LinkedHashMap<Integer, String>() {
        {
            put(40, "XL");
            put(10, "X");
            put(9, "IX");
            put(5, "V");
            put(4, "IV");
            put(1, "I");
        }
    };

    public String convert(int number) {

        String result = "";
        Iterator<Map.Entry<Integer, String>> iterator =
        results.entrySet().iterator();

        while (iterator.hasNext()) {

            Map.Entry<Integer, String> entry = iterator.next();
            while (number >= entry.getKey()) {
                result += entry.getValue();
                number -= entry.getKey();
            }

        }

        return result;
    }
}
```

If to while: pass

The final test.

```java
import junitparams.JUnitParamsRunner;
import junitparams.Parameters;

import org.junit.Test;
import org.junit.runner.RunWith;

import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

@RunWith(JUnitParamsRunner.class)
public class RomanConverterShould {

    @Test
    @Parameters({
            "1, I",
            "2, II",
            "3, III",
            "4, IV",
            "5, V",
            "6, VI",
            "7, VII",
            "8, VIII",
            "9, IX",
            "10, X",
            "40, XL",
            "50, L",
            "90, XC",
            "100, C",
            "400, CD",
            "500, D",
            "900, CM",
            "1000, M",
            "846, DCCCXLVI",
            "1999, MCMXCIX",
            "2008, MMVIII"
    })
    public void convertNumberToRoman(int number, String expected) {
        RomanConverter romanNumeral = new RomanConverter();
        assertThat(romanNumeral.convert(number), is(expected));
    }
}
```

```java
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;

public class RomanConverter {

    private Map<Integer, String> arabicToRomansDictionary =
        new LinkedHashMap<Integer, String>() {
        {
            put(1000, "M");
            put(900, "CM");
            put(500, "D");
            put(400, "CD");
            put(100, "C");
            put(90, "XC");
            put(50, "L");
            put(40, "XL");
            put(10, "X");
            put(9, "IX");
            put(5, "V");
            put(4, "IV");
            put(1, "I");
        }
    };

    public String convert(int number) {

        String result = "";
        Iterator<Map.Entry<Integer, String>> arabicToRomans =
          arabicToRomansDictionary.entrySet().iterator();

        while (arabicToRomans.hasNext()) {

            Map.Entry<Integer, String> arabicToRoman = arabicToRomans.next();
            int arabicNumeral = arabicToRoman.getKey();
            String romanNumeral = arabicToRoman.getValue();

            while (number >= arabicToRoman.getKey()) {
                result += romanNumeral;
                number -= arabicNumeral;
            }
        }

        return result;
    }
}
```

End