CS-107

Mini-Projet 1 - QR Code Generator

Loïc Herman and Kelvin Kappeler

11th November, 2019

This document describes the bonuses we have implemented in our project. These include:
- Revised `addAlignmentPatterns` method to support QR codes of version > 4
- Implemented an `evaluate` method to compute the penalty score of QR codes
- Revised every `QRCodeInfos.java` methods to support QR codes of version > 4 which thus allows QR code generation for all versions and all error correction levels

1 The revised structure

We have done two things to refactor the code into accepting our additions:

1. Add a `public static boolean USE_EXTENSIONS` constant in `MatrixConstruction.java` to use or ignore the potentially automated grader-breaking extensions.
2. Add an additional `Extensions.java` file which has every extensions we have developed.

1.1 `USE_EXTENSIONS`

This `public` constant declared in `MatrixConstruction.java` is also referenced in `DataEncoding.java`. When set to `true`, the QR code construction and data encoding methods which needed to be adapted to support the bonuses will work differently. These said methods use a condition `if (USE_EXTENSIONS) {}` which divides our code into two different sub-mechanisms. This could potentially break an automated grader if they ignore the constant defined above.

1.2 `Extensions.java`

This file is where we have developed all of the methods mentioned above. We will describe their nature and how to use them later on.

This file contains the following methods and classes:
- Revised matrix construction methods
- Revised data encoding methods
- Static `QRCodeInfos` class for better version support
- Static `ErrorCorrectionBlock(s)` classes to hold the data from ISO/IEC 18004

Everything described in this class allows the generator, provided that the extensions are enabled, to generate QR codes with any correction level, for any version, using byte mode encoding (with ISO-8859-1).

2 Extensions

2.1 QR Code evaluation

This method was implemented as described in the given instructions. An overloaded method `public static int[][] renderQRCodeMatrix(int, boolean[])` will return the 2D matrix of the QR code using the best mask found.
We defined four constants `private static final int` `PENALITY_N1`, `PENALITY_N2`, `PENALITY_N3` and `PENALITY_N4`. These constants are not following the given instructions, but are defined according to ISO/IEC 18004, the official specification.
As for the evaluation of the four penalty rules, we also used the ISO/IEC 18004 specification, but it seems that our finder pattern evaluation deviates slightly as we used the sequences given in the instructions. It is possible that our evaluation will suggest a different mask than other generators would.

2.2 Revised matrix construction methods

In the `Extensions.java` file, you will find multiple sections separated by wide comment blocks. At the top of the file, you will find the `public static QRCodeInfos.CorrectionLevel CORRECTION_LEVEL` constant which defines the correction level to be used.
Once the correction level has been set, provided that `USE_EXTENSIONS` has been enabled, the `renderQRCodeMatrix` method will use the revised methods defined in the extensions file. There are no additional requirements to use these methods, except that they allow versions > 4.
JavaDoc comments explain what the methods do in better detail.

2.3 Revised data encoding methods

Again, provided that the two aforementioned variables are correctly set, these methods which are located in the `Extensions.java` file are modified methods to add the data header and footer as well as the error correction blocks as defined in the official specification.
The two methods are described in the corresponding JavaDoc comments.

2.4 Static `QRCodeInfos` class for better version support

As the methods defined in the `QRCodeInfos.java` file are more or less hard coded to only support versions 1 to 4 with error correction low, we needed to create a different class to be able to generate the information required for any version and any error correction level.
This class must be instantiated by giving a version, a mask and a correction level, as defined in the JavaDoc comment. It then has multiple getters which link the appropriate information for the given version and mask. The methods are all corresponding to the ones defined in `QRCodeInfos.java`, with additional methods for error correction blocks for version ≥ 5 and error correction $\geq$ `LOW`, as well as version information for QR codes with version ≥ 7.

2.5 Static `ErrorCorrectionBlock(s)` classes to hold the data from ISO/IEC 18004

As mentioned herein-above, the `Extensions#QRCodeInfos` class uses two different static classes to hold the data for the error correction blocks which is given in an 4D array form from the official specification. Since some versions of QR codes with any error correction level have to have two error correction blocks inscribed, there is one class called `ErrorCorrectionBlocks` which holds one or two `ErrorCorrectionBlock` elements stored in an array defined when these are instantiated from `Extensions#QRCodeInfos`.
These two classes have getters which allows the `Extensions#QRCodeInfos` class' methods to have the different module lengths which are not hard-coded in like the `QRCodeInfos.java` methods.

3 How to use the extensions

In this final section, we will go in detail of how every extension can be enabled, so that includes the constants to change and the parts of the code that need to be changed as well.
The following to lines to edit are always placed at the top of the files.
We will start by editing the `MatrixConstruction.java` file where we will need to change the following line to be `true`:

```
/**
 * (...)
 */
public static boolean USE_EXTENSIONS = false; // set to true
```

We will then move to the `Extensions.java` file where we will edit the following line:

```
/**
 * (...)
 */
public static QRCodeInfos.CorrectionLevel CORRECTION_LEVEL =
    QRCodeInfos.CorrectionLevel.LOW;
```

The values define the error correction level that we want to use, so it can be either of the following four:

1. `QRCodeInfos.CorrectionLevel.LOW`
2. `QRCodeInfos.CorrectionLevel.MEDIUM`
3. `QRCodeInfos.CorrectionLevel.QUARTILE`
4. `QRCodeInfos.CorrectionLevel.HIGH`

And then we head to our main file `Main.java` where we will can change the following constants:

```
public static final String INPUT = "Enter the text here (can also be a QR
    scheme like WIFI)";
// (...)
public static final int VERSION = 10; // Desired version
public static final int MASK = 2;     // Desired mask if not using the
    next change
public static final int SCALING = 12; // Desired scaling for the QR Code
    window
```

And the final change will enable the method to find the best mask for a given input. This is **not** required as mentioned in the listing above. We need to change the following call in `Main.java`:

```
int[][] qrCode =
    MatrixConstruction.renderQRCodeMatrix(VERSION, encodedData, MASK)
```

to be

```
int[][] qrCode =
    MatrixConstruction.renderQRCodeMatrix(VERSION, encodedData);
```

Now all of the extensions will be enabled and should be working for any given input and constants as long as they follow the QR code specification.