Decode of Zxing

For Code93/UPC/EAN Example

Introduction

- Developed by Google
- Open-sourced in Github
  https://github.com/zxing/zxing
- In Java

Supported Formats

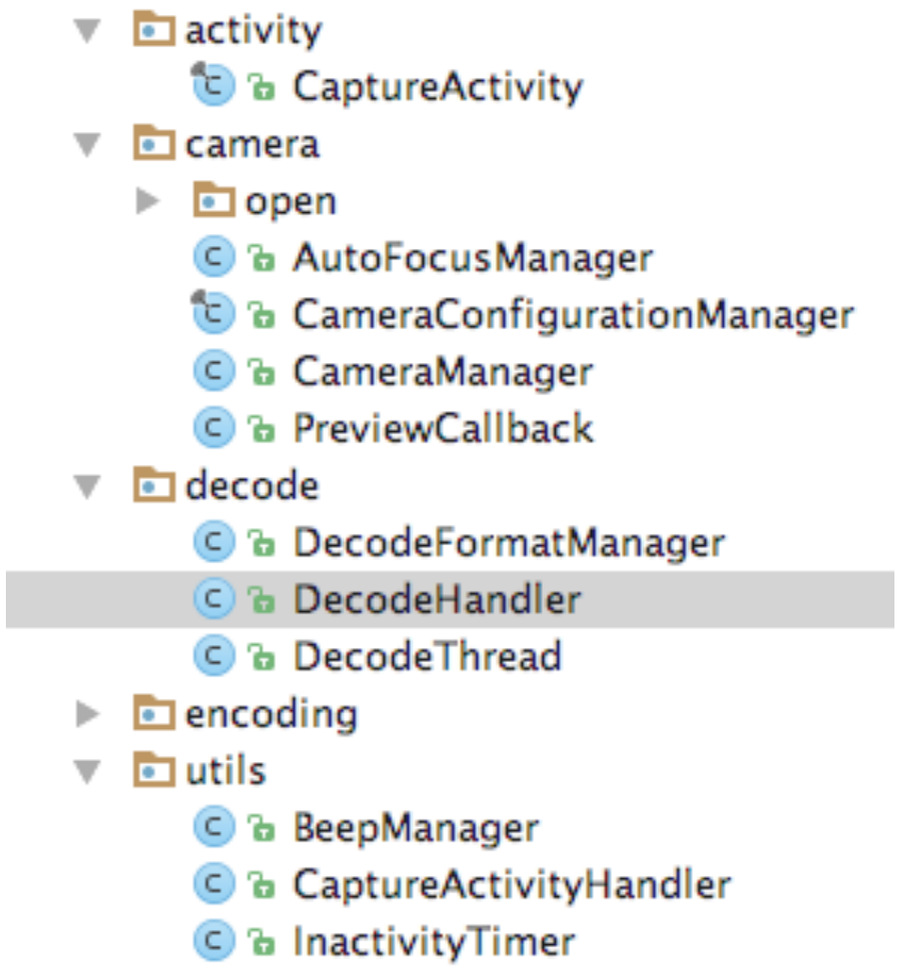
1D product	1D industrial	2D
UPC-A	Code 39	QR Code
UPC-E	Code 93	Data Matrix
EAN-8	Code 128	Aztec (beta)
EAN-13	Codabar	PDF 417 (beta)
	ITF	MaxiCode
	RSS-14	
	RSS-Expanded	

General View

- CaptureActivity -Initialize the scan process

- DecodeHandler -Decode and get the result

- Camera - Camera Control and parameter setting



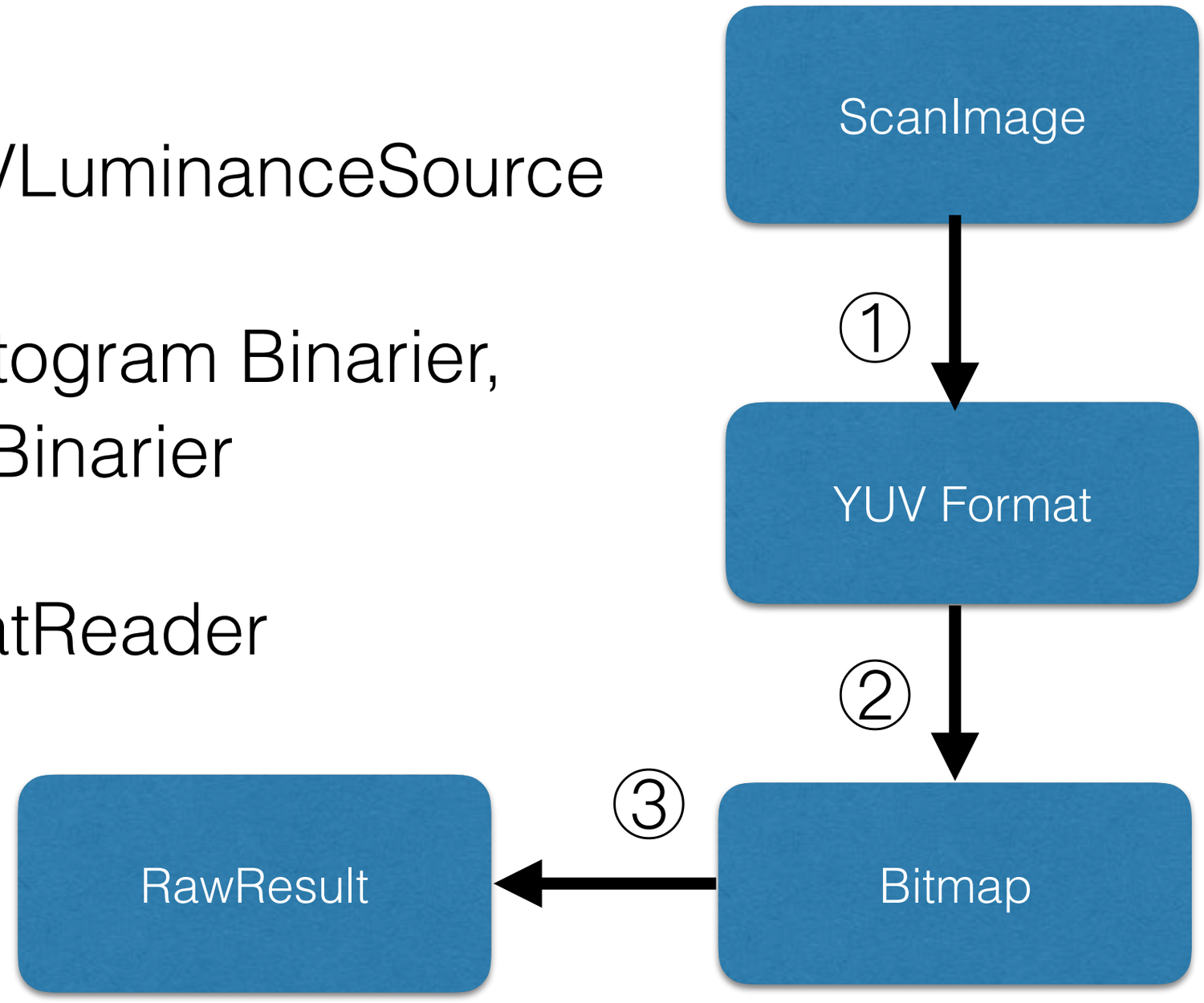
Low-level Decoding

- CodeReader - Called by DecodeHandler to do the decode

- Multiple decoding for different types of code, EAN, QR, Data Matrix, etc.

Detail of Process

- ① PlanarYUVLuminanceSource
- ② GlobalHistogram Binarier, extend from Binarier
- ③ MultiformatReader



MultiFormatReader

- The abstract class for decoding

- Include OneReader(for one-dimensional barcode), QR code

OneDReader.Java

- Encapsulates functionality and implementation that is common to all families of one-dimensional barcodes

- Decode Function: doDecode Method

```
private Result doDecode(BinaryBitmap image,
                        Map<DecodeHintType,?> hints) throws NotFoundException {...}
```

doDecode

- Choose one possible row in barcode
  (1) Start from **middle** of picture (already a bitmap)
  (2) Search from the upward and downward via the row-step(In Zxing, this 1/16) until one row can be decoded

- Apply the specific decodeRow function, if not, try another row.

Example(1)—Code 93

- Extended from Code 39
- Each Character: 9 modules, 3 bars and 3 spaces (named via this)
- From Wikipedia

Code 93 is designed to encode the same 26 upper case letters, 10 digits and 7 special characters as code 39:

```
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
-, ., $, /, +, %, SPACE.
```

Part of Coding Rule

ID	Character	Widths	Binary	ID	Character	Widths	Binary
0	0	131112	100010100	28	S	211122	110101100
1	1	111213	101001000	29	T	211221	110100110
2	2	111312	101000100	30	U	221121	110010110
3	3	111411	101000010	31	V	222111	110011010
4	4	121113	100101000	32	W	112122	101101100
5	5	121212	100100100	33	X	112221	101100110
6	6	121311	100100010	34	Y	122121	100110110
7	7	111114	101010000	35	Z	123111	100111010
8	8	131211	100010010	36	-	121131	100101110
9	9	141111	100001010	37	.	311112	111010100
10	A	211113	110101000	38	SPACE	311211	111010010
11	B	211212	110100100	39	$	321111	111001010
12	C	211311	110100010	40	/	112131	101101110

Code93Reader.Java

```java
public final class Code93Reader extends OneDReader {

  // Note that 'abcd' are dummy characters in place of control characters.
  static final String ALPHABET_STRING = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ-. $/+%abcd*";
  private static final char[] ALPHABET = ALPHABET_STRING.toCharArray();

  /**
   * These represent the encodings of characters, as patterns of wide and narrow bars.
   * The 9 least-significant bits of each int correspond to the pattern of wide and narrow.
   */
  static final int[] CHARACTER_ENCODINGS = {
      0x114, 0x148, 0x144, 0x142, 0x128, 0x124, 0x122, 0x150, 0x112, 0x10A, // 0-9
      0x1A8, 0x1A4, 0x1A2, 0x194, 0x192, 0x18A, 0x168, 0x164, 0x162, 0x134, // A-J
      0x11A, 0x158, 0x14C, 0x146, 0x12C, 0x116, 0x1B4, 0x1B2, 0x1AC, 0x1A6, // K-T
      0x196, 0x19A, 0x16C, 0x166, 0x136, 0x13A, // U-Z
      0x12E, 0x1D4, 0x1D2, 0x1CA, 0x16E, 0x176, 0x1AE, // - - %
      0x126, 0x1DA, 0x1D6, 0x132, 0x15E, // Control chars? $-*
  };
  private static final int ASTERISK_ENCODING = CHARACTER_ENCODINGS[47];
```

Detail of Decode

- Find So-called **Counters** (self-defined, an array presents the number of bar and space) via the black and white pixels

```
protected static void recordPattern(BitArray row,
                                    int start,
                                    int[] counters) throws NotFoundException {...}
```

- Then convert to pattern(defined integer in hex), find the corresponding character.

```
private static int toPattern(int[] counters) {
  int sum = 0;
  for (int counter : counters) {
```

Record Pattern

```
protected static void recordPattern(BitArray row,
                                    int start,
                                    int[] counters) throws NotFoundException {
  int numCounters = counters.length;
  Arrays.fill(counters, 0, numCounters, 0);
  int end = row.getSize();
  if (start >= end) {
    throw NotFoundException.getNotFoundInstance();
  }
  boolean isWhite = !row.get(start);
  int counterPosition = 0;
  int i = start;
  while (i < end) {
    if (row.get(i) != isWhite) {
      counters[counterPosition]++;
    } else {
      if (++counterPosition == numCounters) {
        break;
      } else {
        counters[counterPosition] = 1;
        isWhite = !isWhite;
      }
    }
    i++;
  }
  // If we read fully the last section of pixels and filled up our counters -- or filled
  // the last counter but ran off the side of the image, OK. Otherwise, a problem.
  if (!(counterPosition == numCounters || (counterPosition == numCounters - 1 && i == end))) {
    throw NotFoundException.getNotFoundInstance();
  }
}
```

Note this is general to one-dimension decode!

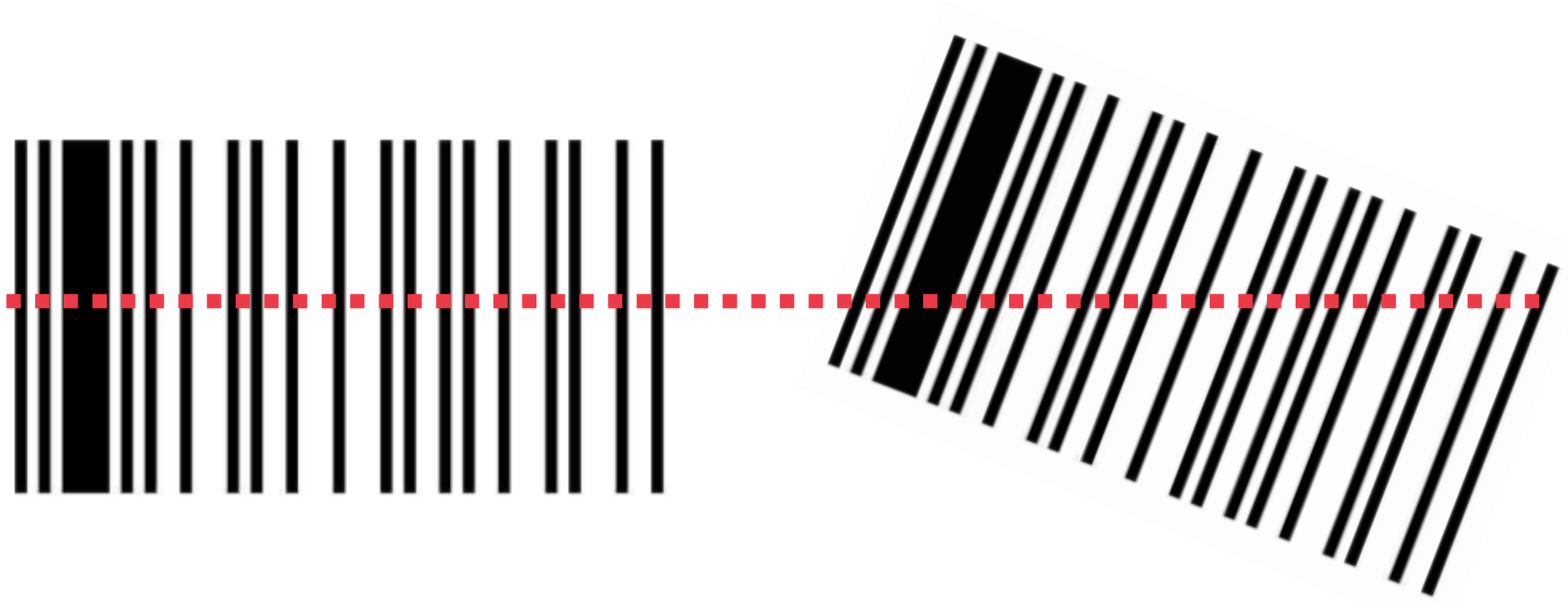
To Pattern

```
private static int toPattern(int[] counters) {
  int sum = 0;
  for (int counter : counters) {
    sum += counter;
  }
  int pattern = 0;
  int max = counters.length;
  for (int i = 0; i < max; i++) {
    int scaled = Math.round(counters[i] * 9.0f / sum);
    if (scaled < 1 || scaled > 4) {
      return -1;
    }
    if ((i & 0x01) == 0) {
      for (int j = 0; j < scaled; j++) {
        pattern = (pattern << 1) | 0x01;
      }
    } else {
      pattern <<= scaled;
    }
  }
  return pattern;
}
```

- Example for Character 1
  1 in Code 93 via **100010100**
  (Binary format)
- 131112 (Bar & Space Width)

Note: (1)This pattern converting is composed on different coding rule!!
(2) Why does the one-dimension code do the rotation for incline?

Continue



- We compare the ratio not the practical width when scanning!