

```
1  package com.chaoticcognitions.aenigma.models.rotors;
2
3  import android.util.Log;
4
5  /**
6   * Class to handle the functionality of the Enigma machine rotors.
7   * @author Dan Cassidy
8   */
9  public class Rotor {
10     public enum Direction {RIGHT_TO_LEFT, LEFT_TO_RIGHT}
11
12     private static final char CHAR_OFFSET = 'A';
13     private final int CHAR_SET_SIZE;
14
15     // Once set, these fields will not be changed. Comprise basic rotor settings. They are stored
16     // inside the Rotor class itself to avoid having an excessive amount of calls to the various
17     // methods of the RotorType enum.
18     private final RotorType rotorType;
19     private final String wiring;
20     private final String reverseWiring;
21     private final String turnoverChars;
22     private final boolean isSteppingRotor;
23     private final boolean isMarkedWithNumbers;
24
25     // User-changeable fields.
26     private char visiblePosition = 'A'; //TODO can change name to something more representative?
27     private char ringSetting = 'A';
28
29     // Automatic field to deal with turnover and stepping.
30     private boolean isAtTurnoverPosition = false;
31     private boolean justStepped = false; //TODO needed?
32     private boolean steppingBuffer = false; //TODO needed?
33
34     /**
35      * 1-parameter constructor.
36      * @param rotorType The type of rotor to construct.
37      */
38     public Rotor(RotorType rotorType) {
39         this.rotorType = rotorType;
40
41         wiring = this.rotorType.wiring();
42         reverseWiring = this.rotorType.reverseWiring();
43         turnoverChars = this.rotorType.turnoverChars();
44         isSteppingRotor = this.rotorType.isSteppingRotor();
45         isMarkedWithNumbers = this.rotorType.isMarkedWithNumbers();
46
47         CHAR_SET_SIZE = wiring.length();
48
49         checkTurnover();
50     }
51
52     // BEGIN GETTERS AND SETTERS -->
53     public RotorType getRotorType() {
54         return rotorType;
55     }
56
57     public String getVisiblePosition() {
58         if (isMarkedWithNumbers) {
59             int visibleNumber = visiblePosition - 'A' + 1;
60             return (visibleNumber < 10 ? "0" + visibleNumber : "" + visibleNumber);
```

```
61         } else
62             return Character.toString(visiblePosition);
63     }
64
65     public void setVisiblePosition(char visiblePosition) throws IllegalArgumentException {
66         if (!isValidChar(visiblePosition))
67             throw new IllegalArgumentException("Invalid position setting.");
68
69         this.visiblePosition = visiblePosition;
70         checkTurnover();
71     }
72
73     public char getRingSetting() {
74         return ringSetting;
75     }
76
77     public void setRingSetting(char ringSetting) throws IllegalArgumentException {
78         if (!isValidChar(ringSetting))
79             throw new IllegalArgumentException("Invalid ring setting.");
80         this.ringSetting = ringSetting;
81     }
82
83     public boolean justStepped() {
84         return justStepped;
85     }
86
87     public boolean isAtTurnoverPosition() {
88         return isAtTurnoverPosition;
89     }
90     // <-- END GETTERS AND SETTERS
91
92     /**
93      * Encodes a character where the input is on the main (right) side of the rotor.
94      * @param inputChar The character to encode.
95      * @return The encoded character.
96      */
97     public char encode(char inputChar, Direction direction) throws IllegalArgumentException {
98         if (!isValidChar(inputChar))
99             throw new IllegalArgumentException("Invalid");
100
101         // Helper code for stepping. //TODO determine if actually needed
102         if (steppingBuffer)
103             steppingBuffer = false;
104         else
105             justStepped = false;
106
107         // Determine the current offset.
108         int offset = ringSetting - visiblePosition;
109         Log.i("Rotor", "Rotor " + rotorType + " [encode(" + direction.toString().charAt(0) + ")]
110         current offset: " + offset + ".");
111
112         // Remove the offset to get the true input character.
113         inputChar -= offset;
114         Log.i("Rotor", "Rotor " + rotorType + " [encode(" + direction.toString().charAt(0) + ")]
115         true input character (pre-normalization): " + inputChar + ".");
116
117         // Normalize the true input character to handle any rollover. (E.g. - A character beyond 'Z'
118         // will get rolled over from the end of the rotor back to the beginning.)
119         inputChar = normalize(inputChar);
120         Log.i("Rotor", "Rotor " + rotorType + " [encode(" + direction.toString().charAt(0) + ")]
```

```
        true input character (post-normalization): " + inputChar + "."));
119
120        // Get the corresponding character that the true input character is wired to on the rotor.
121        char outputChar;
122        if (direction == Direction.RIGHT_TO_LEFT)
123            outputChar = wiring.charAt(inputChar - CHAR_OFFSET);
124        else
125            outputChar = reverseWiring.charAt(inputChar - CHAR_OFFSET);
126        Log.i("Rotor", "Rotor " + rotorType + " [encode(" + direction.toString().charAt(0) + ")]
        true output character: " + outputChar + ".");
127
128        // Add the offset back to the character.
129        outputChar += offset;
130        Log.i("Rotor", "Rotor " + rotorType + " [encode(" + direction.toString().charAt(0) + ")]
        final output character (pre-normalization): " + outputChar + ".");
131
132        // Normalize the offset output character to handle any rollover and get the final output
133        // character.
134        outputChar = normalize(outputChar);
135        Log.i("Rotor", "Rotor " + rotorType + " [encode(" + direction.toString().charAt(0) + ")]
        final output character (post-normalization): " + outputChar + ".");
136
137        // Return the final output character.
138        return outputChar;
139    }
140
141    /**
142     * Steps the rotor.
143     */
144    public void doStep() {
145        // If this rotor does not step, just return, otherwise proceed with stepping.
146        if (!isSteppingRotor)
147            return;
148
149        // TODO: Verify rotor stepping method. <-- Tentatively good.
150        Log.i("Rotor", "Rotor " + rotorType + " stepping from '" + visiblePosition + "' to '" +
151            (visiblePosition == 'Z' ? 'A' : (char)(visiblePosition + 1)) + "'.");
152
153        // Step the rotor and normalize (handle rollover) if needed.
154        visiblePosition = normalize(++visiblePosition);
155
156        // Set some flags to advertise the fact this rotor just stepped. TODO needed?
157        justStepped = true;
158        steppingBuffer = true;
159
160        // Check to see if this rotor is at a turnover position.
161        checkTurnover();
162    }
163
164    /**
165     * Determines whether the rotor is at a turnover position or not.
166     */
167    private void checkTurnover() {
168        isAtTurnoverPosition = (turnoverChars.indexOf(visiblePosition) != -1);
169    }
170
171    /**
172     * Determines whether the argument is a valid character.
173     * @param charToValidate The character to validate.
174     * @return boolean, representing whether the argument is a valid character (true) or not (false).
```

```
175     */
176     private boolean isValidChar(char charToValidate) {
177         return (wiring.indexOf(charToValidate) != -1);
178     }
179
180     /**
181      * Normalize the given character to within the rotor's character set. In other words, handle
182      * the rollover from the end of the character set to the beginning, or from the beginning of the
183      * character set to the end.
184      * @param charToNormalize The character to normalize.
185      * @return char, containing the normalized character.
186      */
187     private char normalize(char charToNormalize) {
188         if (charToNormalize < CHAR_OFFSET)
189             charToNormalize += CHAR_SET_SIZE;
190         else if (charToNormalize >= CHAR_OFFSET + CHAR_SET_SIZE)
191             charToNormalize -= CHAR_SET_SIZE;
192         return charToNormalize;
193     }
194 }
195
```

```
1  /*-----
2  * Author:      Dan Cassidy
3  * Date:        2015-08-04
4  * Assignment:  Project
5  * Source File: RotorType.java
6  * Language:    Java
7  * Course:      CSCI-C 490, Android Programming, MoWe 08:00
8  -----*/
9  package com.chaoticcognitions.aenigma.models.rotors;
10
11 /**
12  * Enum to store the relevant information about the different types of rotors in a single place.
13  * @author Dan Cassidy
14  */
15 public enum RotorType {
16     // Enigma I - German Army and Air Force (Wehrmacht, Luftwaffe)
17     // Stator
18     I_ETW,
19     // Rotors
20     I_I, I_II, I_III, I_IV, I_V,
21     // Reflectors
22     I_UKW_A, I_UKW_B, I_UKW_C,
23
24     // Norway Enigma - Enigma I used postwar by Norway
25     // Stator
26     N_ETW,
27     // Rotors
28     N_I, N_II, N_III, N_IV, N_V,
29     // Reflectors
30     N_UKW,
31
32     // Enigma M3 - German Navy (Kriegsmarine)
33     // Stator
34     M3_ETW,
35     // Rotors
36     M3_I, M3_II, M3_III, M3_IV, M3_V, M3_VI, M3_VII, M3_VIII,
37     // Reflectors
38     M3_UKW_B, M3_UKW_C,
39
40     // Enigma M4 - U-Boot Enigma
41     // Stator
42     M4_ETW,
43     // Rotors
44     M4_I, M4_II, M4_III, M4_IV, M4_V, M4_VI, M4_VII, M4_VIII, M4_BETA, M4_GAMMA,
45     // Reflectors
46     M4_UKW_B, M4_UKW_C,
47
48     // Enigma G - Zählwerk Enigma A28 and G31
49     // Stator
50     G_ETW,
51     // Rotors
52     G_I, G_II, G_III,
53     // Reflectors
54     G_UKW,
55
56     // Enigma D - Commercial Enigma A26
57     // Stator
58     D_ETW,
59     // Rotors
60     D_I, D_II, D_III,
```

```
61 // Reflectors
62 D_UKW,
63
64 // Enigma K - Commercial Enigma A27
65 // Stator
66 K_ETW,
67 // Rotors
68 K_I, K_II, K_III,
69 // Reflectors
70 K_UKW,
71
72 // Swiss K - Swiss Enigma K Variant (Swiss Air Force)
73 // Stator
74 KS_ETW,
75 // Rotors
76 KS_I, KS_II, KS_III,
77 // Reflectors
78 KS_UKW,
79
80 // Enigma KD - Enigma K with UWK-D *** (Rewirable) *** //TODO figure out what to do with the KD
81 // Stator
82 KD_ETW,
83 // Rotors
84 KD_I, KD_II, KD_III,
85 // Reflectors
86 KD_UKW,
87
88 // Railway Enigma - Modified Enigma K
89 // Stator
90 R_ETW,
91 // Rotors
92 R_I, R_II, R_III,
93 // Reflectors
94 R_UKW,
95
96 // Enigma T - Japanese Enigma (Tirpitz)
97 // Stator
98 T_ETW,
99 // Rotors
100 T_I, T_II, T_III, T_IV, T_V, T_VI, T_VII, T_VIII,
101 // Reflectors
102 T_UKW;
103
104 /**
105  * Get the wiring for the rotor based on its type. For instance, 'A' is wired to the first
106  * letter of this string, 'B' is wired to the second letter, and so on.
107  * @return The wiring for the rotor.
108  */
109 public String wiring() {
110     switch (this) {
111         // Stators
112         case I_ETW:
113         case N_ETW:
114         case M3_ETW:
115         case M4_ETW:
116             return "ABCDEFGHGIJKLMNOPQRSTUVWXYZ";
117         case G_ETW:
118         case D_ETW:
119         case K_ETW:
```

```
120     case KS_ETW:
121     case KD_ETW:
122     case R_ETW:
123         return "QWERTZUIOASDFGHJKPYXCVBNML";
124     case T_ETW:
125         return "KZROUQHYAIGBLWVSTDXFPNMCJE";
126
127     // Rotors
128     case I_I:
129     case M3_I:
130     case M4_I:
131         return "EKMFLGDQVZNTOWYHXUSPAIBRCJ";
132     case I_II:
133     case M3_II:
134     case M4_II:
135         return "AJDKSIRUXBLHWTMCQGZNPYFVOE";
136     case I_III:
137     case M3_III:
138     case M4_III:
139         return "BDFHJLCPRTXVZNYEIWGAKMUSQO";
140     case I_IV:
141     case N_IV:
142     case M3_IV:
143     case M4_IV:
144         return "ESOVPPZJAYQUIRHXLNFTGKDCMWB";
145     case I_V:
146     case M3_V:
147     case M4_V:
148         return "VZBRGITYUPSDNHLXAWMJQOFECK";
149     case N_I:
150         return "WTOKASUYVRBXJHQPZEFMDINLG";
151     case N_II:
152         return "GJLPUBSWEMCTQVHXAOZFDRKYNI";
153     case N_III:
154         return "JWFMHNBPUISDYTIHVZGRQLAOEKC";
155     case N_V:
156         return "HEJXQOTZBVFDDASCILWPGYNMURK";
157     case M3_VI:
158     case M4_VI:
159         return "JPGVUUMFYQBENHZRDKASXLICTW";
160     case M3_VII:
161     case M4_VII:
162         return "NZJHGRCXMYSWBOUFAIVLPEKQDT";
163     case M3_VIII:
164     case M4_VIII:
165         return "FKQHTLXOCBJSPDZRAMENIU YGV";
166     case M4_BETA:
167         return "LEYJVCNIXWPBQMDRTAKZGFUHS";
168     case M4_GAMMA:
169         return "FSOKANUERHMBTIYCWLPZXV GJD";
170     case G_I:
171     case D_I:
172     case K_I:
173         return "LPGSZMHAEOQKVXR FYBUTNICJDW";
174     case G_II:
175     case D_II:
176     case K_II:
177         return "SLVGBTFXJQOHEWIRZYAMKPCNDU";
178     case G_III:
179     case D_III:
```

```

180         case K_III:
181             return "CJGDPSHKTURAWZXFMYNQOVBVLE";
182         case KS_I:
183             return "PEZUOHXSCVFM TBGLRINQJWAYDK";
184         case KS_II:
185             return "ZOUESYDKFWPCIQXHMVBLGNJRAT";
186         case KS_III:
187             return "EHRVXGAOBQUSIMZFLYNWKTDPDJC";
188         case KD_I:
189             return "VEZIOJCXKYDUNTWAPLQGBHSFMR";
190         case KD_II:
191             return "HGRBSJZETDLVPMQYCXAKINFUW";
192         case KD_III:
193             return "NWLHXGRBYOJSZDVT PKFQMEUIC";
194         case R_I:
195             return "JGDQOXUSCAMIFRVTPNEWKBLZYH";
196         case R_II:
197             return "NTZPSFBOKMWRCJDIVLAEYUXHGQ";
198         case R_III:
199             return "JVIUBHTCDYAKEQZPOSGXNRMWFL";
200         case T_I:
201             return "KPTYUELOCVGRFQDANJMBSWHZXI";
202         case T_II:
203             return "UPHZLWEQMTDJXCAKSOIGVBYFNR";
204         case T_III:
205             return "QUDLYRFEKONVZAXWHMGPJBSICT";
206         case T_IV:
207             return "CIWTBKXNRESPLYDAGVHQUOJZM";
208         case T_V:
209             return "UAXGISNJBVERDYLFZWTPCKOHMQ";
210         case T_VI:
211             return "XFUZGALVHCNYSWQTD MRBKPIOJ";
212         case T_VII:
213             return "BJVFTXPLNAYOZIKWGDQERUCHSM";
214         case T_VIII:
215             return "YMTPNZHWKODAJXELUQVGC BISFR";
216
217         // Reflectors
218         case I_UKW_A:
219             return "EJMZALYXVBWFCRQUONTSP IKHGD";
220         case I_UKW_B:
221         case M3_UKW_B:
222             return "YRUHQSLDPXNGOKMIEBFZCWVJAT";
223         case I_UKW_C:
224         case M3_UKW_C:
225             return "FVPJIAOYEDRZXWGCTKUQSBNMHL";
226         case N_UKW:
227             return "MOWJYPUXNDSRAIBFVLKZGQCHET";
228         case M4_UKW_B:
229             return "ENKQAUYWJICOPBLMDXZVFTHRGS";
230         case M4_UKW_C:
231             return "RDOBJNTKVEHMLFCWZAXGYIPSUQ";
232         case G_UKW:
233         case D_UKW:
234         case K_UKW:
235         case KS_UKW:
236             return "IMETCGFRAYSQBZXWLHKDVUPOJN";
237         case KD_UKW:
238             return "NSUOMKLIHZFGEADVXWBYCPRQTJ"; // Rewireable!
239         case R_UKW:

```



```
240         return "QYHOGNECVPUZTFDJAXWMKISRBL";
241     case T_UKW:
242         return "GEKPBTAUMOCNILJDXZYFHWVQSR";
243
244     default:
245         return "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
246     }
247 }
248
249 /**
250  * Get the reverse wiring for the different types of rotors.
251  * @return
252  */
253 public String reverseWiring() {
254     String wiring = this.wiring();
255     char[] reverseWiring = new char[wiring.length()];
256
257     final char CHAR_OFFSET = 'A';
258     for (int index = 0; index < wiring.length(); index++)
259         reverseWiring[wiring.charAt(index) - CHAR_OFFSET] = (char)(index + CHAR_OFFSET);
260
261     return new String(reverseWiring);
262 }
263
264 /**
265  * Get the turnover characters for the different types of rotors.
266  */
267 public String turnoverChars() {
268     switch (this) {
269         // Stators
270         case I_ETW:
271         case N_ETW:
272         case M3_ETW:
273         case M4_ETW:
274         case G_ETW:
275         case D_ETW:
276         case K_ETW:
277         case KS_ETW:
278         case KD_ETW:
279         case R_ETW:
280         case T_ETW:
281             return "";
282
283         // Rotors
284         case I_I:
285         case N_I:
286         case M3_I:
287         case M4_I:
288             return "Q";
289         case I_II:
290         case N_II:
291         case M3_II:
292         case M4_II:
293             return "E";
294         case I_III:
295         case N_III:
296         case M3_III:
297         case M4_III:
298             return "V";
299         case I_IV:
```

```
300         case N_IV:
301         case M3_IV:
302         case M4_IV:
303             return "J";
304         case I_V:
305         case N_V:
306         case M3_V:
307         case M4_V:
308             return "Z";
309         case M3_VI:
310         case M4_VI:
311         case M3_VII:
312         case M4_VII:
313         case M3_VIII:
314         case M4_VIII:
315             return "ZM";
316         case M4_BETA:
317         case M4_GAMMA:
318             return "";
319         case G_I:
320             return "SUVWZABCEFGIKLOPQ";
321         case G_II:
322             return "STVYZACDFGHKMNQ";
323         case G_III:
324             return "UWXAEFHKMNR";
325         case D_I:
326         case K_I:
327         case KS_I:
328         case R_III:
329             return "Y";
330         case KS_II:
331         case D_II:
332         case K_II:
333         case R_II:
334             return "E";
335         case D_III:
336         case K_III:
337         case KS_III:
338         case R_I:
339             return "N";
340         case KD_I:
341         case KD_II:
342         case KD_III:
343             return "SUYAEHLNQ";
344         case T_I:
345         case T_III:
346             return "WZEKQ";
347         case T_II:
348         case T_IV:
349             return "WZFLR";
350         case T_V:
351         case T_VII:
352             return "YCFKR";
353         case T_VI:
354         case T_VIII:
355             return "XEIMQ";
356
357         // Reflectors
358         case I_UKW_A:
359         case I_UKW_B:
```

```
360         case I_UKW_C:
361         case N_UKW:
362         case M3_UKW_B:
363         case M3_UKW_C:
364         case M4_UKW_B:
365         case M4_UKW_C:
366         case G_UKW:
367         case D_UKW:
368         case K_UKW:
369         case KS_UKW:
370         case KD_UKW:
371         case R_UKW:
372         case T_UKW:
373             return "";
374
375         default:
376             return "";
377     }
378 }
379
380 /**
381  * Get whether the rotor steps or not based on the type.
382  */
383 public boolean isSteppingRotor() {
384     switch (this) {
385         // Stators
386         case I_ETW:
387         case N_ETW:
388         case M3_ETW:
389         case M4_ETW:
390         case G_ETW:
391         case D_ETW:
392         case K_ETW:
393         case KS_ETW:
394         case KD_ETW:
395         case R_ETW:
396         case T_ETW:
397             return false;
398
399         // Rotors
400         case I_I:
401         case I_II:
402         case I_III:
403         case I_IV:
404         case I_V:
405         case N_I:
406         case N_II:
407         case N_III:
408         case N_IV:
409         case N_V:
410         case M3_I:
411         case M3_II:
412         case M3_III:
413         case M3_IV:
414         case M3_V:
415         case M3_VI:
416         case M3_VII:
417         case M3_VIII:
418         case M4_I:
419         case M4_II:
```

```
420         case M4_III:
421         case M4_IV:
422         case M4_V:
423         case M4_VI:
424         case M4_VII:
425         case M4_VIII:
426         case G_I:
427         case G_II:
428         case G_III:
429         case D_I:
430         case D_II:
431         case D_III:
432         case K_I:
433         case K_II:
434         case K_III:
435         case KS_I:
436         case KS_II:
437         case KS_III:
438         case KD_I:
439         case KD_II:
440         case KD_III:
441         case R_I:
442         case R_II:
443         case R_III:
444         case T_I:
445         case T_II:
446         case T_III:
447         case T_IV:
448         case T_V:
449         case T_VI:
450         case T_VII:
451         case T_VIII:
452             return true;
453         case M4_BETA:
454         case M4_GAMMA:
455             return false;
456
457         // Reflectors
458         case I_UKW_A:
459         case I_UKW_B:
460         case I_UKW_C:
461         case N_UKW:
462         case M3_UKW_B:
463         case M3_UKW_C:
464         case M4_UKW_B:
465         case M4_UKW_C:
466         case D_UKW:
467         case K_UKW:
468         case KS_UKW:
469         case KD_UKW:
470         case R_UKW:
471         case T_UKW:
472             return false;
473         case G_UKW:
474             return true;
475
476         default:
477             return false;
478     }
479 }
```

```
480
481     /**
482     * Get whether the rotor is marked with numbers or not based on the type.
483     */
484     public boolean isMarkedWithNumbers() {
485         switch (this) {
486             // Stators
487             case I_ETW:
488             case N_ETW:
489             case M3_ETW:
490             case M4_ETW:
491             case G_ETW:
492             case D_ETW:
493             case K_ETW:
494             case KS_ETW:
495             case KD_ETW:
496             case R_ETW:
497             case T_ETW:
498                 return false;
499
500             // Rotors
501             case I_I:
502             case I_II:
503             case I_III:
504             case I_IV:
505             case I_V:
506             case N_I:
507             case N_II:
508             case N_III:
509             case N_IV:
510             case N_V:
511                 return true;
512             case M3_I:
513             case M3_II:
514             case M3_III:
515             case M3_IV:
516             case M3_V:
517             case M3_VI:
518             case M3_VII:
519             case M3_VIII:
520             case M4_I:
521             case M4_II:
522             case M4_III:
523             case M4_IV:
524             case M4_V:
525             case M4_VI:
526             case M4_VII:
527             case M4_VIII:
528             case M4_BETA:
529             case M4_GAMMA:
530             case G_I:
531             case G_II:
532             case G_III:
533             case D_I:
534             case D_II:
535             case D_III:
536             case K_I:
537             case K_II:
538             case K_III:
539             case KS_I:
```

```
540         case KS_II:
541         case KS_III:
542         case KD_I:
543         case KD_II:
544         case KD_III:
545         case R_I:
546         case R_II:
547         case R_III:
548         case T_I:
549         case T_II:
550         case T_III:
551         case T_IV:
552         case T_V:
553         case T_VI:
554         case T_VII:
555         case T_VIII:
556             return false;
557
558         // Reflectors
559         case I_UKW_A:
560         case I_UKW_B:
561         case I_UKW_C:
562         case N_UKW:
563         case M3_UKW_B:
564         case M3_UKW_C:
565         case M4_UKW_B:
566         case M4_UKW_C:
567         case G_UKW:
568         case D_UKW:
569         case K_UKW:
570         case KS_UKW:
571         case KD_UKW:
572         case R_UKW:
573         case T_UKW:
574             return false;
575
576         default:
577             return false;
578     }
579 }
580
581 /**
582  * Get whether the rotor is a stator or not.
583  */
584 public boolean isStator() {
585     switch (this) {
586         // Stators
587         case I_ETW:
588         case N_ETW:
589         case M3_ETW:
590         case M4_ETW:
591         case G_ETW:
592         case D_ETW:
593         case K_ETW:
594         case KS_ETW:
595         case KD_ETW:
596         case R_ETW:
597         case T_ETW:
598             return true;
599     }
```

```
600         // Rotors
601         case I_I:
602         case I_II:
603         case I_III:
604         case I_IV:
605         case I_V:
606         case N_I:
607         case N_II:
608         case N_III:
609         case N_IV:
610         case N_V:
611         case M3_I:
612         case M3_II:
613         case M3_III:
614         case M3_IV:
615         case M3_V:
616         case M3_VI:
617         case M3_VII:
618         case M3_VIII:
619         case M4_I:
620         case M4_II:
621         case M4_III:
622         case M4_IV:
623         case M4_V:
624         case M4_VI:
625         case M4_VII:
626         case M4_VIII:
627         case M4_BETA:
628         case M4_GAMMA:
629         case G_I:
630         case G_II:
631         case G_III:
632         case D_I:
633         case D_II:
634         case D_III:
635         case K_I:
636         case K_II:
637         case K_III:
638         case KS_I:
639         case KS_II:
640         case KS_III:
641         case KD_I:
642         case KD_II:
643         case KD_III:
644         case R_I:
645         case R_II:
646         case R_III:
647         case T_I:
648         case T_II:
649         case T_III:
650         case T_IV:
651         case T_V:
652         case T_VI:
653         case T_VII:
654         case T_VIII:
655         return false;
656
657         // Reflectors
658         case I_UKW_A:
659         case I_UKW_B:
```

```
660         case I_UKW_C:
661         case N_UKW:
662         case M3_UKW_B:
663         case M3_UKW_C:
664         case M4_UKW_B:
665         case M4_UKW_C:
666         case G_UKW:
667         case D_UKW:
668         case K_UKW:
669         case KS_UKW:
670         case KD_UKW:
671         case R_UKW:
672         case T_UKW:
673             return false;
674
675         default:
676             return false;
677     }
678 }
679
680 /**
681  * Get whether the rotor is an actual rotor or not.
682  */
683 public boolean isRotor() {
684     switch (this) {
685         // Stators
686         case I_ETW:
687         case N_ETW:
688         case M3_ETW:
689         case M4_ETW:
690         case G_ETW:
691         case D_ETW:
692         case K_ETW:
693         case KS_ETW:
694         case KD_ETW:
695         case R_ETW:
696         case T_ETW:
697             return false;
698
699         // Rotors
700         case I_I:
701         case I_II:
702         case I_III:
703         case I_IV:
704         case I_V:
705         case N_I:
706         case N_II:
707         case N_III:
708         case N_IV:
709         case N_V:
710         case M3_I:
711         case M3_II:
712         case M3_III:
713         case M3_IV:
714         case M3_V:
715         case M3_VI:
716         case M3_VII:
717         case M3_VIII:
718         case M4_I:
719         case M4_II:
```



```
720         case M4_III:
721         case M4_IV:
722         case M4_V:
723         case M4_VI:
724         case M4_VII:
725         case M4_VIII:
726         case M4_BETA:
727         case M4_GAMMA:
728         case G_I:
729         case G_II:
730         case G_III:
731         case D_I:
732         case D_II:
733         case D_III:
734         case K_I:
735         case K_II:
736         case K_III:
737         case KS_I:
738         case KS_II:
739         case KS_III:
740         case KD_I:
741         case KD_II:
742         case KD_III:
743         case R_I:
744         case R_II:
745         case R_III:
746         case T_I:
747         case T_II:
748         case T_III:
749         case T_IV:
750         case T_V:
751         case T_VI:
752         case T_VII:
753         case T_VIII:
754             return true;
755
756         // Reflectors
757         case I_UKW_A:
758         case I_UKW_B:
759         case I_UKW_C:
760         case N_UKW:
761         case M3_UKW_B:
762         case M3_UKW_C:
763         case M4_UKW_B:
764         case M4_UKW_C:
765         case G_UKW:
766         case D_UKW:
767         case K_UKW:
768         case KS_UKW:
769         case KD_UKW:
770         case R_UKW:
771         case T_UKW:
772             return false;
773
774         default:
775             return false;
776     }
777 }
778
779 /**
```

```
780      * Get whether the rotor is a reflector or not.
781      */
782      public boolean isReflector() {
783          switch (this) {
784              // Stators
785              case I_ETW:
786              case N_ETW:
787              case M3_ETW:
788              case M4_ETW:
789              case G_ETW:
790              case D_ETW:
791              case K_ETW:
792              case KS_ETW:
793              case KD_ETW:
794              case R_ETW:
795              case T_ETW:
796              return false;
797
798              // Rotors
799              case I_I:
800              case I_II:
801              case I_III:
802              case I_IV:
803              case I_V:
804              case N_I:
805              case N_II:
806              case N_III:
807              case N_IV:
808              case N_V:
809              case M3_I:
810              case M3_II:
811              case M3_III:
812              case M3_IV:
813              case M3_V:
814              case M3_VI:
815              case M3_VII:
816              case M3_VIII:
817              case M4_I:
818              case M4_II:
819              case M4_III:
820              case M4_IV:
821              case M4_V:
822              case M4_VI:
823              case M4_VII:
824              case M4_VIII:
825              case M4_BETA:
826              case M4_GAMMA:
827              case G_I:
828              case G_II:
829              case G_III:
830              case D_I:
831              case D_II:
832              case D_III:
833              case K_I:
834              case K_II:
835              case K_III:
836              case KS_I:
837              case KS_II:
838              case KS_III:
839              case KD_I:
```

```
840         case KD_II:
841         case KD_III:
842         case R_I:
843         case R_II:
844         case R_III:
845         case T_I:
846         case T_II:
847         case T_III:
848         case T_IV:
849         case T_V:
850         case T_VI:
851         case T_VII:
852         case T_VIII:
853             return false;
854
855         // Reflectors
856         case I_UKW_A:
857         case I_UKW_B:
858         case I_UKW_C:
859         case N_UKW:
860         case M3_UKW_B:
861         case M3_UKW_C:
862         case M4_UKW_B:
863         case M4_UKW_C:
864         case G_UKW:
865         case D_UKW:
866         case K_UKW:
867         case KS_UKW:
868         case KD_UKW:
869         case R_UKW:
870         case T_UKW:
871             return true;
872
873         default:
874             return false;
875     }
876 }
877
878 /**
879  * Returns the string representation of the rotor type.
880  * @return The string representation of the rotor type.
881  */
882 @Override public String toString() {
883     return super.toString().substring(super.toString().indexOf('_') + 1).replace('_', '-');
884 }
885 }
886
```

```
1  //TODO create file comment
2  package com.chaoticcognitions.aenigma.models.pluginboards;
3
4  import android.util.Log;
5
6  /**
7   * Created by Dan on 2015-07-28.
8   */
9  public class Plugboard {
10     private static final char NULLCHAR = '\0';
11     private int numPairs = 0;
12     private char[] plugSettings = new char[26];
13
14     public void addPlugSettings(char char1, char char2) {
15         if (char1 < 'A' || char1 > 'Z' || char2 < 'A' || char2 > 'Z' || char1 == char2)
16             throw new IllegalArgumentException("Invalid characters for plugboard pair.");
17         else if (plugSettings[char1 - 'A'] != NULLCHAR || plugSettings[char2 - 'A'] != NULLCHAR)
18             // At least one of the two characters are paired already.
19             return;
20
21         plugSettings[char1 - 'A'] = char2;
22         plugSettings[char2 - 'A'] = char1;
23         numPairs++;
24     }
25
26     //TODO create method comment
27     public char encode(char inputChar) {
28         Log.i("Plugboard", "Plugboard encoding '" + inputChar + "' to '" + (plugSettings[inputChar
29         - 'A'] == NULLCHAR ? inputChar : plugSettings[inputChar - 'A']) + "'");
30         return (plugSettings[inputChar - 'A'] == NULLCHAR ? inputChar :
31             plugSettings[inputChar - 'A']);
32     }
33 }
```

```
1  //TODO create file comment
2  package com.chaoticcognitions.aenigma.models.machines;
3
4  import com.chaoticcognitions.aenigma.models.pluginboards.Pluginboard;
5  import com.chaoticcognitions.aenigma.models.rotors.Rotor;
6  import com.chaoticcognitions.aenigma.models.rotors.RotorType;
7
8  import static com.chaoticcognitions.aenigma.models.rotors.Rotor.Direction;
9
10 /**
11  * TODO finish class comment
12  * @author Dan Cassidy
13  */
14 public class Machine {
15     public enum RotorPosition {RIGHT, MIDDLE, LEFT, GREEK, REFLECTOR}
16
17     //TODO comment field groupings
18     private final MachineType machineType;
19     private final boolean isEnigmaStepped;
20     private final int numberOfRotors;
21     private final boolean hasVisibleReflector;
22     private final boolean hasPlugboard;
23
24     private final RotorType[] possibleStators;
25     private final RotorType[] possibleRotors;
26     private final RotorType[] possibleReflectors;
27
28     private Plugboard plugboard;
29
30     private Rotor stator;
31     private Rotor[] rotors;
32     private Rotor reflector;
33
34     //TODO create method comment
35     public Machine(MachineType machineType) {
36         this.machineType = machineType;
37
38         isEnigmaStepped = this.machineType.isEnigmaStepped();
39         numberOfRotors = this.machineType.numberOfRotors();
40         hasVisibleReflector = this.machineType.hasVisibleReflector();
41         hasPlugboard = this.machineType.hasPlugboard();
42         possibleStators = this.machineType.possibleStators();
43         possibleRotors = this.machineType.possibleRotors();
44         possibleReflectors = this.machineType.possibleReflectors();
45
46         plugboard = new Plugboard();
47
48         rotors = new Rotor[numberOfRotors];
49     }
50
51     // BEGIN GETTERS AND SETTERS -->
52     public MachineType getMachineType() {
53         return machineType;
54     }
55
56     public void setStator(RotorType statorType) {
57         if (!isValidStator(statorType))
58             throw new IllegalArgumentException("Invalid stator type.");
59         this.stator = new Rotor(statorType);
60     }
```

```
61
62     public void setReflector(RotorType reflectorType) {
63         if (!isValidReflector(reflectorType))
64             throw new IllegalArgumentException("Invalid reflector type.");
65         reflector = new Rotor(reflectorType);
66     }
67
68     public void setRotor(RotorType rotorType, RotorPosition position) {
69         if (!isValidRotor(rotorType) || !isValidPosition(position))
70             throw new IllegalArgumentException("Invalid rotor type or position.");
71         rotors[position.ordinal()] = new Rotor(rotorType);
72     }
73
74     public void setPlugboardPairs(String plugPairs) {
75         for (int index = 0; index < plugPairs.length() && index + 1 < plugPairs.length(); index += 2)
76             plugboard.addPlugSettings(plugPairs.charAt(index), plugPairs.charAt(index + 1));
77     }
78
79     public void setRingSetting(char ringSetting, RotorPosition position) {
80         if (!isValidPosition(position))
81             throw new IllegalArgumentException("Invalid position.");
82         if (position == RotorPosition.REFLECTOR)
83             reflector.setRingSetting(ringSetting);
84         else
85             rotors[position.ordinal()].setRingSetting(ringSetting);
86     }
87
88     public void setVisiblePosition(char visiblePosition, RotorPosition position) {
89         if (!isValidPosition(position))
90             throw new IllegalArgumentException("Invalid position.");
91         if (position == RotorPosition.REFLECTOR)
92             reflector.setVisiblePosition(visiblePosition);
93         else
94             rotors[position.ordinal()].setVisiblePosition(visiblePosition);
95     }
96     // <-- END GETTERS AND SETTERS
97
98     //TODO create method comment
99     public char encode(char inputChar) {
100         // step
101         doStep();
102         // encode plugboard
103         if (hasPlugboard)
104             inputChar = plugboard.encode(inputChar);
105         // encode stator
106         inputChar = stator.encode(inputChar, Direction.RIGHT_TO_LEFT);
107         // encode rotor array
108         for (Rotor rotor : rotors)
109             inputChar = rotor.encode(inputChar, Direction.RIGHT_TO_LEFT);
110         // encode reflector
111         inputChar = reflector.encode(inputChar, Direction.RIGHT_TO_LEFT);
112         // encode rotor array (reverse)
113         for (int index = numberOfRotors - 1; index >= 0; index--)
114             inputChar = rotors[index].encode(inputChar, Direction.LEFT_TO_RIGHT);
115         // encode stator (reverse)
116         inputChar = stator.encode(inputChar, Direction.LEFT_TO_RIGHT);
117         // encode plugboard (reverse)
118         if (hasPlugboard)
119             inputChar = plugboard.encode(inputChar);
120         return inputChar;
```

```
121     }
122
123     //TODO create method comment
124     public String encode(String inputString) {
125         String toReturn = "";
126         for (char inputChar : inputString.toCharArray())
127             toReturn += encode(inputChar);
128         return toReturn;
129     }
130
131     //TODO create method comment
132     private void doStep() {
133         //TODO see if this method can be optimized at all
134         if (isEnigmaStepped) {
135             if (rotors[RotorPosition.RIGHT.ordinal()].isAtTurnoverPosition()) {
136                 // normal stepping
137                 if (rotors[RotorPosition.MIDDLE.ordinal()].isAtTurnoverPosition()) {
138                     if (rotors[RotorPosition.LEFT.ordinal()].isAtTurnoverPosition()) {
139                         reflector.doStep();
140                     }
141                     rotors[RotorPosition.LEFT.ordinal()].doStep();
142                 }
143                 rotors[RotorPosition.MIDDLE.ordinal()].doStep();
144             } else {
145                 // double stepping (?)
146                 if (rotors[RotorPosition.MIDDLE.ordinal()].isAtTurnoverPosition() &&
147                     rotors[RotorPosition.MIDDLE.ordinal()].justStepped()) {
148                     rotors[RotorPosition.LEFT.ordinal()].doStep();
149                     rotors[RotorPosition.MIDDLE.ordinal()].doStep();
150                 }
151             }
152             rotors[RotorPosition.RIGHT.ordinal()].doStep();
153         } else {
154             if (rotors[RotorPosition.RIGHT.ordinal()].isAtTurnoverPosition()) {
155                 if (rotors[RotorPosition.MIDDLE.ordinal()].isAtTurnoverPosition()) {
156                     if (rotors[RotorPosition.LEFT.ordinal()].isAtTurnoverPosition()) {
157                         reflector.doStep();
158                     }
159                     rotors[RotorPosition.LEFT.ordinal()].doStep();
160                 }
161                 rotors[RotorPosition.MIDDLE.ordinal()].doStep();
162             }
163             rotors[RotorPosition.RIGHT.ordinal()].doStep();
164         }
165     }
166
167     //TODO create method comment
168     @Override public String toString() {
169         String toReturn = "";
170         for (Rotor rotor : rotors)
171             toReturn = rotor.getVisiblePosition() + (toReturn.isEmpty() ? "" : " ") + toReturn;
172         if (hasVisibleReflector)
173             toReturn = reflector.getVisiblePosition() + " " + toReturn;
174         return toReturn;
175     }
176
177     //TODO create method comment
178     private boolean isValidStator(RotorType statorToValidate) {
179         for (RotorType stator : possibleStators)
180             if (statorToValidate == stator)
```

```
181         return true;
182
183     return false;
184 }
185
186 //TODO create method comment
187 private boolean isValidRotor(RotorType rotorToValidate) {
188     for (RotorType rotor : possibleRotors)
189         if (rotorToValidate == rotor)
190             return true;
191
192     return false;
193 }
194
195 //TODO create method comment
196 private boolean isValidReflector(RotorType reflectorToValidate) {
197     for (RotorType reflector : possibleReflectors)
198         if (reflectorToValidate == reflector)
199             return true;
200
201     return false;
202 }
203
204 //TODO create method comment
205 private boolean isValidPosition(RotorPosition positionToValidate) {
206     return !(positionToValidate == RotorPosition.GREEK && numberOfRotors != 4);
207 }
208
209 //TODO create method comment
210 private boolean isReady() {
211     for (Rotor rotor : rotors)
212         if (rotor == null)
213             return false;
214     return (stator != null && reflector != null && plugboard != null);
215 }
216 }
217
```



```
1  /*-----
2  * Author:      Dan Cassidy
3  * Date:        2015-08-04
4  * Assignment:  Project
5  * Source File: MachineType.java
6  * Language:    Java
7  * Course:      CSCI-C 490, Android Programming, MoWe 08:00
8  -----*/
9  package com.chaoticcognitions.aenigma.models.machines;
10
11  import com.chaoticcognitions.aenigma.models.rotors.RotorType;
12
13  /**
14   * Enum to store the relevant information about the different types of Enigma machines in a single
15   * place.
16   * @author Dan Cassidy
17   */
18  public enum MachineType {
19      ENIGMA_I,
20      NORWAY_ENIGMA,
21      ENIGMA_M3,
22      ENIGMA_M4,
23      ENIGMA_G,
24      ENIGMA_D,
25      ENIGMA_K,
26      SWISS_K,
27      ENIGMA_KD,
28      RAILWAY_ENIGMA,
29      ENIGMA_T;
30
31  /**
32   * Gets a list of possible stators for the given machine type.
33   * @return The possible stators for the given machine type.
34   */
35  public RotorType[] possibleStators() {
36      switch (this) {
37          case ENIGMA_I:
38              return new RotorType[]{RotorType.I_ETW};
39          case NORWAY_ENIGMA:
40              return new RotorType[]{RotorType.N_ETW};
41          case ENIGMA_M3:
42              return new RotorType[]{RotorType.M3_ETW};
43          case ENIGMA_M4:
44              return new RotorType[]{RotorType.M4_ETW};
45          case ENIGMA_G:
46              return new RotorType[]{RotorType.G_ETW};
47          case ENIGMA_D:
48              return new RotorType[]{RotorType.D_ETW};
49          case ENIGMA_K:
50              return new RotorType[]{RotorType.K_ETW};
51          case SWISS_K:
52              return new RotorType[]{RotorType.KS_ETW};
53          case ENIGMA_KD:
54              return new RotorType[]{RotorType.KD_ETW};
55          case RAILWAY_ENIGMA:
56              return new RotorType[]{RotorType.R_ETW};
57          case ENIGMA_T:
58              return new RotorType[]{RotorType.T_ETW};
59
60          default:
```

```
61         return new RotorType[]{};
62     }
63 }
64
65 /**
66  * Gets a list of possible rotors for the given machine type.
67  * @return The possible rotors for the given machine type.
68  */
69 public RotorType[] possibleRotors() {
70     switch (this) {
71         case ENIGMA_I:
72             return new RotorType[]{RotorType.I_I, RotorType.I_II, RotorType.I_III,
73                                     RotorType.I_IV, RotorType.I_V};
74         case NORWAY_ENIGMA:
75             return new RotorType[]{RotorType.N_I, RotorType.N_II, RotorType.N_III,
76                                     RotorType.N_IV, RotorType.N_V};
77         case ENIGMA_M3:
78             return new RotorType[]{RotorType.M3_I, RotorType.M3_II, RotorType.M3_III,
79                                     RotorType.M3_IV, RotorType.M3_V, RotorType.M3_VI, RotorType.M3_VII,
80                                     RotorType.M3_VIII};
81         case ENIGMA_M4:
82             return new RotorType[]{RotorType.M4_I, RotorType.M4_II, RotorType.M4_III,
83                                     RotorType.M4_IV, RotorType.M4_V, RotorType.M4_VI, RotorType.M4_VII,
84                                     RotorType.M4_VIII, RotorType.M4_BETA, RotorType.M4_GAMMA};
85         case ENIGMA_G:
86             return new RotorType[]{RotorType.G_I, RotorType.G_II, RotorType.G_III};
87         case ENIGMA_D:
88             return new RotorType[]{RotorType.D_I, RotorType.D_II, RotorType.D_III};
89         case ENIGMA_K:
90             return new RotorType[]{RotorType.K_I, RotorType.K_II, RotorType.K_III};
91         case SWISS_K:
92             return new RotorType[]{RotorType.KS_I, RotorType.KS_II, RotorType.KS_III};
93         case ENIGMA_KD:
94             return new RotorType[]{RotorType.KD_I, RotorType.KD_II, RotorType.KD_III};
95         case RAILWAY_ENIGMA:
96             return new RotorType[]{RotorType.R_I, RotorType.R_II, RotorType.R_III};
97         case ENIGMA_T:
98             return new RotorType[]{RotorType.T_I, RotorType.T_II, RotorType.T_III,
99                                     RotorType.T_IV, RotorType.T_V, RotorType.T_VI, RotorType.T_VII, RotorType.T_VIII};
100     default:
101         return new RotorType[]{};
102     }
103 }
104
105 /**
106  * Gets a list of possible reflectors for the given machine type.
107  * @return The possible reflectors for the given machine type.
108  */
109 public RotorType[] possibleReflectors(){
110     switch (this) {
111         case ENIGMA_I:
112             return new RotorType[]{RotorType.I_UKW_A, RotorType.I_UKW_B, RotorType.I_UKW_C};
113         case NORWAY_ENIGMA:
114             return new RotorType[]{RotorType.N_UKW};
115         case ENIGMA_M3:
116             return new RotorType[]{RotorType.M3_UKW_B, RotorType.M3_UKW_C};
117         case ENIGMA_M4:
118             return new RotorType[]{RotorType.M4_UKW_B, RotorType.M4_UKW_C};
119         case ENIGMA_G:
```

```
114         return new RotorType[] {RotorType.G_UKW};
115     case ENIGMA_D:
116         return new RotorType[] {RotorType.D_UKW};
117     case ENIGMA_K:
118         return new RotorType[] {RotorType.K_UKW};
119     case SWISS_K:
120         return new RotorType[] {RotorType.KS_UKW};
121     case ENIGMA_KD:
122         return new RotorType[] {RotorType.KD_UKW};
123     case RAILWAY_ENIGMA:
124         return new RotorType[] {RotorType.R_UKW};
125     case ENIGMA_T:
126         return new RotorType[] {RotorType.T_UKW};
127
128     default:
129         return new RotorType[] {};
130     }
131 }
132
133 /**
134  * Gets whether the machine is Enigma stepped or not based on the type.
135  * @return The possible statots for the given machine type.
136  */
137 public boolean isEnigmaStepped() {
138     switch (this) {
139         case ENIGMA_I:
140         case NORWAY_ENIGMA:
141         case ENIGMA_M3:
142         case ENIGMA_M4:
143         case ENIGMA_D:
144         case ENIGMA_K:
145         case SWISS_K:
146         case ENIGMA_KD:
147         case RAILWAY_ENIGMA:
148         case ENIGMA_T:
149             return true;
150         case ENIGMA_G:
151             return false;
152
153         default:
154             return true;
155     }
156 }
157
158 /**
159  * Gets whether the machine's reflector is visible or not based on the type.
160  * @return Whether the machine's reflector is visible.
161  */
162 public boolean hasVisibleReflector() {
163     switch (this) {
164         case ENIGMA_I:
165         case NORWAY_ENIGMA:
166         case ENIGMA_M3:
167         case ENIGMA_M4:
168             return false;
169         case ENIGMA_G:
170         case ENIGMA_D:
171         case ENIGMA_K:
172         case SWISS_K:
173         case ENIGMA_KD:
```

```
174         case RAILWAY_ENIGMA:
175         case ENIGMA_T:
176             return true;
177
178         default:
179             return false;
180     }
181 }
182
183 //TODO create method comment
184 public boolean hasPlugboard() {
185     switch (this) {
186         case ENIGMA_I:
187         case NORWAY_ENIGMA:
188         case ENIGMA_M3:
189         case ENIGMA_M4:
190             return true;
191         case ENIGMA_G:
192         case ENIGMA_D:
193         case ENIGMA_K:
194         case SWISS_K:
195         case ENIGMA_KD:
196         case RAILWAY_ENIGMA:
197         case ENIGMA_T:
198             return false;
199
200         default:
201             return false;
202     }
203 }
204
205 /**
206  * Gets the number of rotors a machine has based on its type. Note that this is only describing
207  * the number of actual rotors and does not include the stator or reflector.
208  * @return The number of rotors the machine has.
209  */
210 public int numberOfRotors() {
211     switch (this) {
212         case ENIGMA_I:
213         case NORWAY_ENIGMA:
214         case ENIGMA_M3:
215         case ENIGMA_G:
216         case ENIGMA_D:
217         case ENIGMA_K:
218         case SWISS_K:
219         case ENIGMA_KD:
220         case RAILWAY_ENIGMA:
221         case ENIGMA_T:
222             return 3;
223         case ENIGMA_M4:
224             return 4;
225
226         default:
227             return 3;
228     }
229 }
230
231 /**
232  * Returns the string representation of the machine type.
233  * @return The string representation of the machine type.
```

```
234     */
235     @Override public String toString() {
236         switch (this) {
237             case ENIGMA_I:
238                 return "Enigma I";
239             case NORWAY_ENIGMA:
240                 return "Norway Enigma";
241             case ENIGMA_M3:
242                 return "Enigma M3";
243             case ENIGMA_M4:
244                 return "Enigma M4";
245             case ENIGMA_G:
246                 return "Enigma G";
247             case ENIGMA_D:
248                 return "Enigma D";
249             case ENIGMA_K:
250                 return "Enigma K";
251             case SWISS_K:
252                 return "Swiss-K";
253             case ENIGMA_KD:
254                 return "Enigma KD";
255             case RAILWAY_ENIGMA:
256                 return "Railway Enigma";
257             case ENIGMA_T:
258                 return "Enigma T";
259
260             default:
261                 return "Unknown";
262         }
263     }
264 }
265
```