```java
package com.chaoticcognitions.aenigma.models.rotors;

import android.util.Log;

/**
 * Class to handle the functionality of the Enigma machine rotors.
 * @author Dan Cassidy
 */
public class Rotor {
    public enum Direction {RIGHT_TO_LEFT, LEFT_TO_RIGHT}

    private static final char CHAR_OFFSET = 'A';
    private final int CHAR_SET_SIZE;

    // Once set, these fields will not be changed. Comprise basic rotor settings. They are stored
    // inside the Rotor class itself to avoid having an excessive amount of calls to the various
    // methods of the RotorType enum.
    private final RotorType rotorType;
    private final String wiring;
    private final String reverseWiring;
    private final String turnoverChars;
    private final boolean isSteppingRotor;
    private final boolean isMarkedWithNumbers;

    // User-changeable fields.
    private char visiblePosition = 'A'; //TODO can change name to something more representative?
    private char ringSetting = 'A';

    // Automatic field to deal with turnover and stepping.
    private boolean isAtTurnoverPosition = false;
    private boolean justStepped = false; //TODO needed?
    private boolean steppingBuffer = false; //TODO needed?

    /**
     * 1-parameter constructor.
     * @param rotorType The type of rotor to construct.
     */
    public Rotor(RotorType rotorType) {
        this.rotorType = rotorType;

        wiring = this.rotorType.wiring();
        reverseWiring = this.rotorType.reverseWiring();
        turnoverChars = this.rotorType.turnoverChars();
        isSteppingRotor = this.rotorType.isSteppingRotor();
        isMarkedWithNumbers = this.rotorType.isMarkedWithNumbers();

        CHAR_SET_SIZE = wiring.length();

        checkTurnover();
    }

    // BEGIN GETTERS AND SETTERS -->
    public RotorType getRotorType() {
        return rotorType;
    }

    public String getVisiblePosition() {
        if (isMarkedWithNumbers) {
            int visibleNumber = visiblePosition - 'A' + 1;
            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) + ")]
            current offset: " + offset + ".");
110
111         // Remove the offset to get the true input character.
112         inputChar -= offset;
113         Log.i("Rotor", "Rotor " + rotorType + " [encode(" + direction.toString().charAt(0) + ")]
            true input character (pre-normalization): " + inputChar + ".");
114
115         // Normalize the true input character to handle any rollover. (E.g. - A character beyond 'Z'
116         // will get rolled over from the end of the rotor back to the beginning.)
117         inputChar = normalize(inputChar);
118         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
```