```java
//TODO create file comment
package com.chaoticcognitions.aenigma.models.machines;

import com.chaoticcognitions.aenigma.models.plugboards.Plugboard;
import com.chaoticcognitions.aenigma.models.rotors.Rotor;
import com.chaoticcognitions.aenigma.models.rotors.RotorType;

import static com.chaoticcognitions.aenigma.models.rotors.Rotor.Direction;

/**
 * TODO finish class comment
 * @author Dan Cassidy
 */
public class Machine {
    public enum RotorPosition {RIGHT, MIDDLE, LEFT, GREEK, REFLECTOR}

    //TODO comment field groupings
    private final MachineType machineType;
    private final boolean isEnigmaStepped;
    private final int numberOfRotors;
    private final boolean hasVisibleReflector;
    private final boolean hasPlugboard;

    private final RotorType[] possibleStators;
    private final RotorType[] possibleRotors;
    private final RotorType[] possibleReflectors;

    private Plugboard plugboard;

    private Rotor stator;
    private Rotor[] rotors;
    private Rotor reflector;

    //TODO create method comment
    public Machine(MachineType machineType) {
        this.machineType = machineType;

        isEnigmaStepped = this.machineType.isEnigmaStepped();
        numberOfRotors = this.machineType.numberOfRotors();
        hasVisibleReflector = this.machineType.hasVisibleReflector();
        hasPlugboard = this.machineType.hasPlugboard();
        possibleStators = this.machineType.possibleStators();
        possibleRotors = this.machineType.possibleRotors();
        possibleReflectors = this.machineType.possibleReflectors();

        plugboard = new Plugboard();

        rotors = new Rotor[numberOfRotors];
    }

    // BEGIN GETTERS AND SETTERS -->
    public MachineType getMachineType() {
        return machineType;
    }

    public void setStator(RotorType statorType) {
        if (!isValidStator(statorType))
            throw new IllegalArgumentException("Invalid stator type.");
        this.stator = new Rotor(statorType);
    }
```

```
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)
```

```
                return true;

            return false;
        }

        //TODO create method comment
        private boolean isValidRotor(RotorType rotorToValidate) {
            for (RotorType rotor : possibleRotors)
                if (rotorToValidate == rotor)
                    return true;

            return false;
        }

        //TODO create method comment
        private boolean isValidReflector(RotorType reflectorToValidate) {
            for (RotorType reflector : possibleReflectors)
                if (reflectorToValidate == reflector)
                    return true;

            return false;
        }

        //TODO create method comment
        private boolean isValidPosition(RotorPosition positionToValidate) {
            return !(positionToValidate == RotorPosition.GREEK && numberOfRotors != 4);
        }

        //TODO create method comment
        private boolean isReady() {
            for (Rotor rotor : rotors)
                if (rotor == null)
                    return false;
            return (stator != null && reflector != null && plugboard != null);
        }
    }

```