```java
package EntireMachine;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;


/**
 * Rotors selectable, can currently put multiple of same rotor in different slots. All rotors start at position 1.

 * Program now runs as accurately as version 0.1.0 but is now split into multiple classes.
 *
 * Release as EnigmaMachine Version 0.3.0
 *
 * Once some time has been spent to add proper comments, increment version by 0.0.1
 *
 * Implementaion and release plan still to work on:
 *
 * 0.5.0 remove selected rotors from other dropdown and re-add as available if removed.
 *
 * 0.6.0 implement selectable start position
 *
 * 0.7.0 change position from number to character representation
 *
 * 0.8.0 add "export to text" button, add "clear all/reset" button
 *
 * 0.8.5 add top bar menu with "about" or similar. eg. a model tag
 *
 * 0.8.6 finish adding notes if not already complete.
 *
 * 1.0.0 package as a full release version 1
 *
 * 1.1.0 design plugboard and add single cable/connector
 *
 * 1.2.0 add show/hide plugboard button
 *
 * 1.3.0 implement full ten "cables"
 *
 * 1.4.0 add "remove all" button for removing all cables.
 *
 * 1.5.0 add and implement additional reflectors
 *
 * 2.0.0 package as full release version 2
 *
 * 2.0.1 plan additional beautification and functions including:
 * - add drag/drop funtion to connector plugboard
 * - add drag/drop function for rotors
 * - add rotate function for rotors
 * 3.0.0 release as full release version 3
 *
 * Author: Michael Legg
 */


/*****************************************************************************************************************/
```

```
56
57 public class EnigmaMachine {
58
59     private EnigmaGUI gui = new EnigmaGUI();
60     private KeyPressHandler kHandler = new KeyPressHandler();
61     private RSlot1Handler r1Handler = new RSlot1Handler();
62     private RSlot2Handler r2Handler = new RSlot2Handler();
63     private RSlot3Handler r3Handler = new RSlot3Handler();
64     private DisplayUpdate dManager = new DisplayUpdate(gui);
65
66     private static String userInput = String.valueOf('\0');
67
68     private static String[] possibleRotors = { String.valueOf('\0'),"I", "II", "III", "IV",
   "V" };
69
70     private static String EncodedTranslate = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
71
72     private static String EncodedReflector1 = "EJMZALYXVBWFCRQUONTSPIKHGD";
73     private static String EncodedReflector2 = "YRUHQSLDPXNGOKMIEBFZCWVJAT";
74     private static String EncodedReflector3 = "FVPJIAOYEDRZXWGCTKUQSBNMHL";
75
76     private static int index, ascii;
77     private static int tempNumHold, testNum, selectedRotor1, selectedRotor2, selectedRotor3;
78     private static char tempCharHold;
79     private static char pressedKey, encodedKey;
80
81     private static char[][] RotorSlot1 = new char[2][26];
82     private static char[][] RotorSlot2 = new char[2][26];
83     private static char[][] RotorSlot3 = new char[2][26];
84     private static char[][] Reflector = new char[2][26];
85
86     private static boolean[] RotorAvailable = new boolean[5];
87
88     private static int Rotor1Pos = 1;
89     private static int Rotor2Pos = 1;
90     private static int Rotor3Pos = 1;
91
92
93
94   /*************************************************************************************
   ************************************************************/
95
96
97     /**
98      * Launch the application.
99      */
100    public static void main(String[] args) {
101
102        new EnigmaMachine();
103
104    }
105
106
107  /*************************************************************************************
   ************************************************************/
```

```
108
109
110     /**
111      * Create the application.
112      */
113     public EnigmaMachine() {
114
115         for(index = 0; index < 5; index++) {
116             RotorAvailable[index] = true;
117         }
118
119         for(index = 0; index < 26; index++) {
120             Reflector[0][index] = EncodedTranslate.charAt(index);
121             Reflector[1][index] = EncodedReflector1.charAt(index);
122         }
123
124         gui.EnigmaGUI(kHandler, r1Handler, r2Handler, r3Handler);
125         dManager.updateDisplay();
126
127     }
128
129
130 /*************************************************************************************************
    ***************************************************************/
131
132     /********************************** Set Rotors ***************************************/
133
134     //****************Rotor1 setup*********************//
135
136     public static void SetRotor1(int rotorSelect) {
137
138         if (rotorSelect == 1) {
139             for(index = 0; index < 26; index++) {
140                 RotorSlot1[0][index] = EncodedTranslate.charAt(index);
141                 RotorSlot1[1][index] = Rotors.Rotor1.ReturnCharAt(index);
142             }
143             RotorAvailable[0] = false;
144         }
145         else if (rotorSelect == 2) {
146             for(index = 0; index < 26; index++) {
147                 RotorSlot1[0][index] = EncodedTranslate.charAt(index);
148                 RotorSlot1[1][index] = Rotors.Rotor2.ReturnCharAt(index);
149             }
150             RotorAvailable[1] = false;
151         }
152         else if (rotorSelect == 3) {
153             for(index = 0; index < 26; index++) {
154                 RotorSlot1[0][index] = EncodedTranslate.charAt(index);
155                 RotorSlot1[1][index] = Rotors.Rotor3.ReturnCharAt(index);
156             }
157             RotorAvailable[2] = false;
158         }
159         else if (rotorSelect == 4) {
160             for(index = 0; index < 26; index++) {
161                 RotorSlot1[0][index] = EncodedTranslate.charAt(index);
162                 RotorSlot1[1][index] = Rotors.Rotor4.ReturnCharAt(index);
```

```
163             }
164             RotorAvailable[3] = false;
165         }
166         else if (rotorSelect == 5) {
167             for(index = 0; index < 26; index++) {
168                 RotorSlot1[0][index] = EncodedTranslate.charAt(index);
169                 RotorSlot1[1][index] = Rotors.Rotor5.ReturnCharAt(index);
170             }
171             RotorAvailable[4] = false;
172         }
173         else {
174             System.out.println("Invalid rotorSelect available");
175         }
176 
177         return;
178     }
179 
180     //****************Rotor2 setup*********************//
181 
182     public static void SetRotor2(int rotorSelect) {
183 
184         if (rotorSelect == 1) {
185             for(index = 0; index < 26; index++) {
186                 RotorSlot2[0][index] = EncodedTranslate.charAt(index);
187                 RotorSlot2[1][index] = Rotors.Rotor1.ReturnCharAt(index);
188             }
189             RotorAvailable[0] = false;
190         }
191         else if (rotorSelect == 2) {
192             for(index = 0; index < 26; index++) {
193                 RotorSlot2[0][index] = EncodedTranslate.charAt(index);
194                 RotorSlot2[1][index] = Rotors.Rotor2.ReturnCharAt(index);
195             }
196             RotorAvailable[1] = false;
197         }
198         else if (rotorSelect == 3) {
199             for(index = 0; index < 26; index++) {
200                 RotorSlot2[0][index] = EncodedTranslate.charAt(index);
201                 RotorSlot2[1][index] = Rotors.Rotor3.ReturnCharAt(index);
202             }
203             RotorAvailable[2] = false;
204         }
205         else if (rotorSelect == 4) {
206             for(index = 0; index < 26; index++) {
207                 RotorSlot2[0][index] = EncodedTranslate.charAt(index);
208                 RotorSlot2[1][index] = Rotors.Rotor4.ReturnCharAt(index);
209             }
210             RotorAvailable[3] = false;
211         }
212         else if (rotorSelect == 5) {
213             for(index = 0; index < 26; index++) {
214                 RotorSlot2[0][index] = EncodedTranslate.charAt(index);
215                 RotorSlot2[1][index] = Rotors.Rotor5.ReturnCharAt(index);
216             }
217             RotorAvailable[4] = false;
218         }
219         else {
```

```
220             System.out.println("Invalid rotorSelect available");
221         }
222 
223         return;
224     }
225 
226 
227         //****************Rotor3 setup*********************//
228 
229     public static void SetRotor3(int rotorSelect) {
230 
231         if (rotorSelect == 1) {
232             for(index = 0; index < 26; index++) {
233                 RotorSlot3[0][index] = EncodedTranslate.charAt(index);
234                 RotorSlot3[1][index] = Rotors.Rotor1.ReturnCharAt(index);
235             }
236             RotorAvailable[0] = false;
237         }
238         else if (rotorSelect == 2) {
239             for(index = 0; index < 26; index++) {
240                 RotorSlot3[0][index] = EncodedTranslate.charAt(index);
241                 RotorSlot3[1][index] = Rotors.Rotor2.ReturnCharAt(index);
242             }
243             RotorAvailable[1] = false;
244         }
245         else if (rotorSelect == 3) {
246             for(index = 0; index < 26; index++) {
247                 RotorSlot3[0][index] = EncodedTranslate.charAt(index);
248                 RotorSlot3[1][index] = Rotors.Rotor3.ReturnCharAt(index);
249             }
250             RotorAvailable[2] = false;
251         }
252         else if (rotorSelect == 4) {
253             for(index = 0; index < 26; index++) {
254                 RotorSlot3[0][index] = EncodedTranslate.charAt(index);
255                 RotorSlot3[1][index] = Rotors.Rotor4.ReturnCharAt(index);
256             }
257             RotorAvailable[3] = false;
258         }
259         else if (rotorSelect == 5) {
260             for(index = 0; index < 26; index++) {
261                 RotorSlot3[0][index] = EncodedTranslate.charAt(index);
262                 RotorSlot3[1][index] = Rotors.Rotor5.ReturnCharAt(index);
263             }
264             RotorAvailable[4] = false;
265         }
266         else {
267             System.out.println("Invalid rotorSelect available");
268         }
269 
270         return;
271     }
272 
273 
274     //end of Set Rotors sector
275 
276 
```

```
    /*****************************************************************************************
    ***************************************************************/
277
278
279     public void PositionSet(int selectedRotor, int setPosition) {
280         int rotate = 0;
281
282 //      //*********Set Rotor1 Position******************//
283 //      if (selectedRotor == 1) {
284 //          for (rotate = 0; rotate < setPosition-1; rotate++) {
285 //              for(testNum = 0; testNum <=1; testNum++) {
286 //                  tempCharHold = RotorSlot1[testNum][0];
287 //
288 //                  for(index = 1; index <= 25; index++) {
289 //                      RotorSlot1[testNum][index-1] = RotorSlot1[testNum][index];
290 //                  }
291 //                  RotorSlot1[testNum][index-1] = tempCharHold;
292 //              }
293 //          }
294 //          Rotor1Pos = setPosition;
295 //      }
296 //      else{};
297 //
298 //
299 //      //*********Set Rotor2 Position******************//
300 //      if (selectedRotor == 2) {
301 //          for (rotate = 0; rotate < setPosition-1; rotate++) {
302 //              for(testNum = 0; testNum <=1; testNum++) {
303 //                  tempCharHold = RotorSlot1[testNum][0];
304 //
305 //                  for(index = 1; index <= 25; index++) {
306 //                      RotorSlot2[testNum][index-1] = RotorSlot2[testNum][index];
307 //                  }
308 //                  RotorSlot2[testNum][index-1] = tempCharHold;
309 //              }
310 //          }
311 //          Rotor2Pos = setPosition;
312 //      }
313 //      else{};
314 //
315 //
316 //      //*********Set Rotor3 Position******************//
317 //      if (selectedRotor == 3) {
318 //          for (rotate = 0; rotate < setPosition-1; rotate++) {
319 //              for(testNum = 0; testNum <=1; testNum++) {
320 //                  tempCharHold = RotorSlot1[testNum][0];
321 //
322 //                  for(index = 1; index <= 25; index++) {
323 //                      RotorSlot3[testNum][index-1] = RotorSlot3[testNum][index];
324 //                  }
325 //                  RotorSlot3[testNum][index-1] = tempCharHold;
326 //              }
327 //          }
328 //          Rotor3Pos = setPosition;
329 //      }
330 //      else{};
331 //
```

```
332 //      return;
333     }//end of PositionSet()
334 
335 
336 /*****************************************************************************************
    ***************************************************************/
337 
338 
339     public static void PassKeyPress() {
340 
341         encodedKey = '\0';
342 
343         if (ascii >= 65 && ascii <= 90) {
344             EncodeChar();
345                 userInput += String.valueOf(encodedKey);
346         }
347         else {}
348 
349         CheckNewLine();
350 
351     return;
352     }
353 
354 
355 /*****************************************************************************************
    ***************************************************************/
356 
357 
358     public static void EncodeChar() {
359 
360         ascii = (ascii - 65);
361 
362         //Key Input
363         char input = pressedKey;
364 
365 
366         //Input to Rotor1
367         input = RotorSlot1[0][ascii];
368         //Through Rotor1
369         index = 0;
370         while(RotorSlot1[1][index] != input) {
371             index++;
372         }
373 
374 
375         //Input to Rotor2
376         input = RotorSlot2[0][index];
377         //Through Rotor2
378         index = 0;
379         while(RotorSlot2[1][index] != input) {
380             index++;
381         }
382 
383 
384         //Input to Rotor3
```

```
385         input = RotorSlot3[0][index];
386         //Through Rotor3
387         index = 0;
388         while(RotorSlot3[1][index] != input) {
389             index++;
390         }
391 
392 
393         //Input to Reflector
394         input = Reflector[0][index];
395         //Through Reflector
396         input = Reflector[1][index];
397         index = 0;
398         while(Reflector[0][index] != input) {
399             index++;
400         }
401 
402 
403         //Input to Rotor3
404         input = RotorSlot3[1][index];
405         //Through Rotor3
406         index = 0;
407         while(RotorSlot3[0][index] != input) {
408             index++;
409         }
410 
411 
412         //Input to Rotor2
413         input = RotorSlot2[1][index];
414         //Through Rotor2
415         index = 0;
416         while(RotorSlot2[0][index] != input) {
417             index++;
418         }
419 
420 
421         //Input to Rotor1
422         input = RotorSlot1[1][index];
423         //Through Rotor1
424         index = 0;
425         while(RotorSlot1[0][index] != input) {
426             index++;
427         }
428 
429 
430         //Rotor1 to Output
431         encodedKey = (char)(index+65);
432 
433 
434         CycleRotors();
435 
436         return;
437     }
438 
439 
440 /**********************************************************************************************
```

```
**************************************************************/


    public static void CycleRotors() {

        //Rotate Rotor 1;
        for(testNum = 0; testNum <=1; testNum++) {
            tempCharHold = RotorSlot1[testNum][0];

            for(index = 1; index <= 25; index++) {
                RotorSlot1[testNum][index-1] = RotorSlot1[testNum][index];
            }
            RotorSlot1[testNum][index-1] = tempCharHold;
        }
        Rotor1Pos++;

        //Check if Rotor 1 is back at start
        if (Rotor1Pos > 26) {
            Rotor1Pos = 1;

            //Rotate Rotor 2
            for(testNum = 0; testNum <=1; testNum++) {
                tempCharHold = RotorSlot2[testNum][0];

                for(index = 1; index <= 25; index++) {
                    RotorSlot2[testNum][index-1] = RotorSlot2[testNum][index];
                }
                RotorSlot2[testNum][index-1] = tempCharHold;
            }
            Rotor2Pos++;
        }

        //Check if Rotor 2 is back at start
        if (Rotor2Pos > 26) {
            Rotor2Pos = 1;

            //Rotate Rotor 3
            for(testNum = 0; testNum <=1; testNum++) {
                tempCharHold = RotorSlot3[testNum][0];

                for(index = 1; index <= 25; index++) {
                    RotorSlot3[testNum][index-1] = RotorSlot3[testNum][index];
                }
                RotorSlot3[testNum][index-1] = tempCharHold;
            }
            Rotor3Pos++;
        }

        if (Rotor3Pos > 26) {
            Rotor3Pos = 1;
        }

        return;
    }
```

```
    /*****************************************************************************************
    ****************************************************************/
497
498
499     public static void CheckNewLine() {
500         testNum = 0;
501         tempNumHold = userInput.length();
502         testNum = tempNumHold % 45;
503
504         if (tempNumHold > 0 && testNum == 0) {
505             userInput += "\r\n";
506         }
507
508         return;
509     }
510
511
512
    /*****************************************************************************************
    ****************************************************************/
513
514     public static String SendUserInput() {
515         return userInput;
516     }
517
518     public static int SendRotor1Pos() {
519         return Rotor1Pos;
520     }
521
522     public static int SendRotor2Pos() {
523         return Rotor2Pos;
524     }
525
526     public static int SendRotor3Pos() {
527         return Rotor3Pos;
528     }
529
530     public static int SendRotor1Selections() {
531         return selectedRotor1;
532     }
533
534     public static int SendRotor2Selections() {
535         return selectedRotor2;
536     }
537
538     public static int SendRotor3Selections() {
539         return selectedRotor3;
540     }
541
542     public static String availableRotor(int i) {
543         if (RotorAvailable[i] == true) {
544             return possibleRotors[i+1];
545         }
546         else {
547             return "false";
548         }
549     }
```

```
550
551
552
    /*****************************************************************************************
    ********************************************************/
553
554         public class KeyPressHandler implements KeyListener {
555
556             public void keyPressed(KeyEvent e) {
557                 ascii = e.getKeyCode();
558                 pressedKey = e.getKeyChar();
559                 pressedKey = Character.toUpperCase(pressedKey);
560                 EnigmaMachine.PassKeyPress();
561                 dManager.updateDisplay();
562
563             }
564
565             public void keyTyped(KeyEvent e) {
566             }
567             public void keyReleased(KeyEvent e) {
568             }
569
570         } //end of class KeyPressHandler
571
572
573         public class RSlot1Handler implements ActionListener {
574             public void actionPerformed(ActionEvent e) {
575                 String r1 = e.getSource().toString();
576                 r1 = RotorActionSubString.makeSubStr(r1);
577                 switch (r1) {
578
579                     case "I":
580                         selectedRotor1 = 1;
581                         EnigmaMachine.SetRotor1(1);
582                         break;
583
584                     case "II":
585                         selectedRotor1 = 2;
586                         EnigmaMachine.SetRotor1(2);
587                         break;
588
589                     case "III":
590                         selectedRotor1 = 3;
591                         EnigmaMachine.SetRotor1(3);
592                         break;
593
594                     case "IV":
595                         selectedRotor1 = 4;
596                         EnigmaMachine.SetRotor1(4);
597                         break;
598
599                     case "V":
600                         selectedRotor1 = 5;
601                         EnigmaMachine.SetRotor1(5);
602                         break;
603
604                     default:
```

```
605                     System.out.println("RSlot1 Error");
606                 }
607             }
608         } //end of class RotorHandler
609 
610         public class RSlot2Handler implements ActionListener {
611             public void actionPerformed(ActionEvent e) {
612                 String r2 = e.getSource().toString();
613                 r2 = RotorActionSubString.makeSubStr(r2);
614                 switch (r2) {
615                     case "I":
616                         selectedRotor2 = 1;
617                         EnigmaMachine.SetRotor2(1);
618                         break;
619 
620                     case "II":
621                         selectedRotor2 = 2;
622                         EnigmaMachine.SetRotor2(2);
623                         break;
624 
625                     case "III":
626                         selectedRotor2 = 3;
627                         EnigmaMachine.SetRotor2(3);
628                         break;
629 
630                     case "IV":
631                         selectedRotor2 = 4;
632                         EnigmaMachine.SetRotor2(4);
633                         break;
634 
635                     case "V":
636                         selectedRotor2 = 5;
637                         EnigmaMachine.SetRotor2(5);
638                         break;
639                     default:
640                         System.out.println("RSlot2 Error");
641                 }
642             }
643         } //end of class RotorHandler
644 
645         public class RSlot3Handler implements ActionListener {
646             public void actionPerformed(ActionEvent e) {
647                 String r3 = e.getSource().toString();
648                 r3 = RotorActionSubString.makeSubStr(r3);
649                 switch (r3) {
650                     case "I":
651                         selectedRotor3 = 1;
652                         EnigmaMachine.SetRotor3(1);
653                         break;
654 
655                     case "II":
656                         selectedRotor3 = 2;
657                         EnigmaMachine.SetRotor2(2);
658                         break;
659 
660                     case "III":
661                         selectedRotor3 = 3;
```

```
662                         EnigmaMachine.SetRotor3(3);
663                         break;
664
665                     case "IV":
666                         selectedRotor3 = 4;
667                         EnigmaMachine.SetRotor3(4);
668                         break;
669
670                     case "V":
671                         selectedRotor3 = 5;
672                         EnigmaMachine.SetRotor3(5);
673                         break;
674
675                     default:
676                         System.out.println("RSlot3 Error");
677                 }
678             }
679         } //end of class RotorHandler
680
681         public static class RotorActionSubString {
682
683             public static String makeSubStr(String e) {
684
685                 int subStrStart = e.lastIndexOf("selectedItemReminder=") + 21;
686                 int subStrEnd = e.indexOf(']', subStrStart);
687                 e = e.substring(subStrStart, subStrEnd);
688
689                 return e;
690             }
691
692         }
693
694
695 } //end of class EnigmaMachine
696
697
698
699
700
701
702
703
704
705
706
707
708
709
```