```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Data;
 4  using System.Text;
 5  using System.Windows;
 6
 7  /*
 8   * Title:   UtilityValidator
 9   * Author:  Paul McKillop
10   * Date:    March 2020
11   * Purpose: Apply rules to items
12   */
13
14  namespace SignatureGeneratorV
15  {
16      public class UtilityValidator
17      {
18          //-- Rules as module-wide variables
19          static int strengthRuleMinimum = UtilityZGlobals.LengthRule();
20
21          /// <summary>
22          /// Check meets length standard
23          /// </summary>
24          /// <param name="myString"></param>
25          /// <returns></returns>
26          public static bool LengthRuleCheck(string myString)
27          {
28              return myString.Length >= strengthRuleMinimum;
29          }
30
31
32          #region CharactersAllValid
33          /// <summary>
34          /// Check if the characters in user string are all valid
35          /// </summary>
36          /// <param name="stringToCheck"></param>
37          /// <returns></returns>
38          public static bool CharactersAllValid(string stringToCheck)
39          {
40              bool tempBool = true;
41
42              List<string> validCharacterCodes = Lists.ValidCharacterCodes();
43
44              stringToCheck.ToCharArray();
45
46              int stringLength = stringToCheck.Length;
47
48              for (int i = 0; i < stringLength; i++)
49              {
50                  if (!Lists.StringFound(validCharacterCodes, ((int)
                      stringToCheck[i]).ToString()))
51                  {
52                      tempBool = false;
53                  }
54              }
55
```

```
56              return tempBool;
57          }
58          #endregion
59
60          #region GetInvalidCharacter
61          /// <summary>
62          /// Error infomation of invalid character userstring data
63          /// </summary>
64          /// <param name="stringToCheck"></param>
65          /// <returns></returns>
66          public static InvalidCharacter GetInvalidCharacter(string
              stringToCheck)
67          {
68
69              var invalidCharacter = new InvalidCharacter();
70
71              List<string> validCharacterCodes = Lists.ValidCharacterCodes();
72
73              stringToCheck.ToCharArray();
74
75              int stringLength = stringToCheck.Length;
76
77              for (int i = 0; i < stringLength; i++)
78              {
79                  if (!Lists.StringFound(validCharacterCodes, ((int)
                      stringToCheck[i]).ToString()))
80                  {
81                      invalidCharacter.Character = stringToCheck[i].ToString();
82                      invalidCharacter.Position = i + 1;
83                      break;
84                  }
85              }
86
87              return invalidCharacter;
88          }
89          #endregion
90
91
92
93          public static int CurrentLetterScore(string letterToCheck)
94          {
95              string path = (@"E:\ascii.txt");
96              var tempScore = 0;
97
98              DataTable characterData = new DataTable();
99
100             characterData = UtilityCharacterDb.GetCharacterData(path);
101
102             foreach (DataRow row in characterData.Rows)
103             {
104                 var currentCharacter = new Character
105                 {
106                     Code = row.Field<string>(0),
107                     Score = row.Field<string>(1)
108                 };
109
```

```
110                 if (currentCharacter.Code == letterToCheck)
111                 {
112                     tempScore = Int32.Parse(currentCharacter.Score);
113                 }
114 
115             }
116 
117             return tempScore;
118 
119         }
120 
121 
122         public static int WholeStringScore(string userstring)
123         {
124             var tempScore = 0;
125             var stringLength = userstring.Length;
126 
127             userstring.ToCharArray();
128 
129             for (int i = 0; i < stringLength; i++)
130             {
131                 tempScore += UtilityValidator.CurrentLetterScore(((int)
                      userstring[i]).ToString());
132             }
133 
134             return tempScore;
135         }
136 
137         /// <summary>
138         /// Grade of string as full string
139         /// </summary>
140         /// <param name="strengthScore"></param>
141         /// <returns></returns>
142         public static string StrengthGradeLong(int strengthScore)
143         {
144             string outcome;
145 
146             switch (strengthScore)
147             {
148                 case int n when (n <= 7):
149                     outcome = "Unnacceptable";
150                     break;
151                 case int n when (n >= 8 && n <= 10):
152                     outcome = "Weak";
153                     break;
154                 case int n when (n >= 11 && n <= 16):
155                     outcome = "Medium";
156                     break;
157                 case int n when (n >= 17):
158                     outcome = "Strong";
159                     break;
160                 default:
161                     outcome = "Invalid";
162                     break;
163             }
164 
```

```
165
166             return outcome;
167         }
168
169
170         /// <summary>
171         /// Individual character score
172         /// </summary>
173         /// <param name="characterCode"></param>
174         /// <returns></returns>
175         public static int CharacterScore(string characterCode)
176         {
177             int charScore = 0;
178
179             List<Character> characters = Lists.Characters();
180             foreach (Character character in characters)
181             {
182                 if (character.Code == characterCode)
183                 {
184                     charScore = Int32.Parse(character.Score);
185                     return charScore;
186                 }
187             }
188
189             return charScore;
190         }
191     }
192 }
193
```