```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Windows;
5  using System.Windows.Controls;
6
7  /*
8   * Title:   PageActivity Code behind
9   * Author:  Paul McKillop
10  * Date:    01 Januray 2020
11  * Purpose: Code behind for functionality
12  */
13
14 namespace GymTracking
15 {
16     /// <summary>
17     /// Interaction logic for PageActivity.xaml
18     /// </summary>
19     ///
20
21
22     public partial class PageActivity : Page
23     {
24
25         #region 01 Handler variables
26         //-- handler variables
27         internal static string selectedMachine = "";
28         internal static string selectedLevel = "";
29         internal bool isWeighted = false;
30         internal int durationRecorded = 0;
31         internal int usageRate = 0;
32         internal double usedRunningTotal = 0;
33         internal int totalMinutesOfExercise = 0;
34         //-- structures to hold data for diplay showing activities recorded
35         internal List<Activity> recordedActivities = new List<Activity>();
36         internal List<string> activitiesToDisplay = new List<string>();
37
38         //-- Object to handle person data after passed
39         internal static Person person = new Person();
40
41         //- Object to hold the Activity data
42         internal static Summary summary = new Summary();
43
44         //-- Track the number of activities
45         internal static int activitiesRecorded = 0;
46         #endregion
47
48         #region 02 03 Constructors
49         //**************************************
50         //-- DEFAULT constructor
51         //**************************************
52         //-- include the Person data passed from PagePerson as a parameter
53         //-- for the Constructor of the page
54         public PageActivity(Person personPassed)
55         {
56             InitializeComponent();
```

```
57              //-- assign the passed data to the module wide variable
58              person = personPassed;
59
60              //-- assign the person to the summary opbject
61              summary.SessionPerson = person;
62
63              //-- Hide the inclined controls by defailt unless Treadmill
                  selected
64              InclinedCheckBoxLabel.Visibility = Visibility.Hidden;
65              InclinedCheckBox.Visibility = Visibility.Hidden;
66
67          }
68
69          //-- alternative constructor for when loaded without a person object
              available
70          public PageActivity()
71          {
72              InitializeComponent();
73
74              //-- Hide the inclined controls by defailt unless Treadmill
                  selected
75              InclinedCheckBoxLabel.Visibility = Visibility.Hidden;
76              InclinedCheckBox.Visibility = Visibility.Hidden;
77
78              //-- set up list headers
79              var headers = "Machine" + "\t" + "Used" + "\t" + "used";
80              //-- add to list
81              activitiesToDisplay.Add(headers);
82          }
83          #endregion
84
85          #region 04 Loaded
86          //-- Executes on page loaded
87          private void Page_Loaded(object sender, RoutedEventArgs e)
88          {
89              //-- confirm the person data has been received
90              MessageBox.Show("Person data received for " + person.PersonName);
91          }
92          #endregion
93
94
95          #region Navigation Buttons
96          private void BackButton_Click(object sender, RoutedEventArgs e)
97          {
98              var pagePerson = new PagePerson();
99              this.NavigationService.Navigate(pagePerson);
100         }
101
102         private void PageSummaryButton_Click(object sender, RoutedEventArgs e)
103         {
104             //-- finalise the summary object
105             summary.NumberOfActivities = recordedActivities.Count;
106             summary.MinutesOfExercise = totalMinutesOfExercise;
107             summary.TotalUsed = (int)usedRunningTotal;
108
109
```

```
110             //-- Use navigation service to go to page
111             //-- Pass the summary object
112             var pageSummary = new PageSummary(summary);
113             this.NavigationService.Navigate(pageSummary);
114         }
115         #endregion
116
117
118         #region Machines Combo handler methods
119         //-- Load the data when the control is loaded to the form
120         private void MachinesCombo_Loaded(object sender, RoutedEventArgs e)
121         {
122             var combo = sender as ComboBox;
123             combo.ItemsSource = Machines();
124             combo.SelectedIndex = 0;
125         }
126
127         //-- Update the module wide variable when item selected
128         private void MachinesCombo_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
129         {
130             var selectedMachineCombo = sender as ComboBox;
131             selectedMachine = selectedMachineCombo.SelectedItem as string;
132
133             //-- Control the visibility of the checkbox control label and
                  checkbox
134             if (selectedMachine == "Treadmill")
135             {
136                 InclinedCheckBoxLabel.Visibility = Visibility.Visible;
137                 InclinedCheckBox.Visibility = Visibility.Visible;
138             }
139             else
140             {
141                 InclinedCheckBoxLabel.Visibility = Visibility.Hidden;
142                 InclinedCheckBox.Visibility = Visibility.Hidden;
143             }
144
145         }
146         #endregion
147
148         #region Levels Combo handler methods
149         private void LevelsCombo_Loaded(object sender, RoutedEventArgs e)
150         {
151             var combo = sender as ComboBox;
152             combo.ItemsSource = Levels();
153             combo.SelectedIndex = 0;
154         }
155
156         private void LevelsCombo_SelectionChanged(object sender,
              SelectionChangedEventArgs e)
157         {
158             var selectedLevelCombo = sender as ComboBox;
159             selectedLevel = selectedLevelCombo.SelectedItem as string;
160
161             ////-- DEBUG
162             //MessageBox.Show("Selected level is " + selectedLevel);
```

```
163         }
164         #endregion
165
166         //-- Process the data on the form.
167         #region Data processing methods
168         private void AddActivityButton_Click(object sender, RoutedEventArgs e)
169         {
170             //-- Activity objhect to hold data
171             Activity currentActivity = new Activity();
172
173             //-- assign by harvesting form data
174             currentActivity = HarvestForm;
175
176             //-- increase the count of activities
177             activitiesRecorded++;
178
179             //-- Increase the totalMinutes
180             totalMinutesOfExercise += currentActivity.Duration;
181
182             //-- Check that no more than 6 activities recorded
183             if (activitiesRecorded <= 6)
184             {
185                 //-- Add the current activity to the summary object
186                 //summary.Activities.Add(currentActivity);
187                 recordedActivities.Add(currentActivity);
188
189                 //-- Message box to show provcess has worked
190                 //MessageBox.Show("Number of recorded activities: " +
                      recordedActivities.Count.ToString());
191
192                 //-- show that activity is added
193                 CountOfActivitiesTextBlock.Text = activitiesRecorded.ToString
                      ();
194
195                 //-- Reset text box
196                 DurationTextBox.Text = "";
197
198                 //-- Manage display string, first this activity
199                 //-- to be added to listr for display
200                 var listString = MakeSingleDisplayString(currentActivity);
201                 activitiesToDisplay.Add(listString);
202
203                 //-- refresh and display list of activities
204                 //-- by making one string from the whole list
205                 ActivityListTextBlock.Text = MakeWholeDisplayString
                      (activitiesToDisplay);
206             }
207             else // Message that limit exceeded
208             {
209                 MessageBox.Show("The maximum number of activities is 6");
210             }
211         }
212
213         private Activity HarvestForm
214         {
215             get
```

```
216             {
217                 //-- See if weighted
218                 isWeighted = InclinedCheckBox.IsChecked ?? false;
219 
220                 //-- Handler variables
221                 double durationFractionOfHour = 0;
222                 double usedInActivity = 0;
223                 float weightingFactor = 1.11F;
224 
225                 //-- Error check the duration.
226                 //-- First, is there a value?
227                 //-- second, does value meet rules of 5 - 60 minutes?
228                 //-- Use a nested 'if' construct
229                 if (!string.IsNullOrEmpty(DurationTextBox.Text))
230                 {
231                     var durationRecordedToCheck = Convert.ToInt32
                    (DurationTextBox.Text);
232                     //-- check it meets length rule
233                     if (ActivityValidation.ActivityDurationValid
                    (durationRecordedToCheck))
234                     {
235                         durationRecorded = Convert.ToInt32
                    (DurationTextBox.Text);
236                     }
237                     else //-- Outcome for value outsdide the rule 5 - 60
                    minutes
238                     {
239                         MessageBox.Show(" The activity duration must be
                    between 5 and 60 mminutes");
240                     }
241                 }
242                 else //-- Outcome for empty duration text box
243                 {
244                     MessageBox.Show("You must enter a duration for the
                    activity");
245                 }
246 
247                 //-- convert minutes to a fraction of an hour.
248                 durationFractionOfHour = FractionOfHour(durationRecorded);
249 
250                 //-- Get the Rate for combination of Machine and Level
251                 usageRate = MachineDataDb.GetRate(selectedMachine,
                  selectedLevel);
252 
253                 //-- If the weighted/inlined check box true, usage is
                  increased by 11%
254                 //-- Else, straughtforward multiplication of Rate per Hour *
                  Fraction of hour recorded
255                 if (isWeighted)
256                 {
257                     usedInActivity = (usageRate * durationFractionOfHour) *
                    weightingFactor;
258                 }
259                 else
260                 {
261                     usedInActivity = usageRate * durationFractionOfHour;
```

```
262                 }
263
264                 //-- Add new data
265                 usedRunningTotal += usedInActivity;
266
267                 //-- Assign values to the object to be returned
268                 Activity tempActivity = new Activity
269                 {
270                     MachineName = selectedMachine,
271                     Weighted = isWeighted,
272                     Level = selectedLevel,
273                     Duration = durationRecorded,
274                     Used = usedInActivity
275                 };
276
277                 StringBuilder sb = new StringBuilder();
278
279                 sb.Append("Person name: ").AppendLine
                      (summary.SessionPerson.PersonName);
280                 sb.Append("Person age: ").AppendLine
                      (summary.SessionPerson.Age.ToString());
281                 sb.Append("Person weight: ").AppendLine
                      (summary.SessionPerson.Weight.ToString());
282                 sb.AppendLine();
283                 sb.AppendLine(selectedMachine);
284                 sb.AppendLine(isWeighted.ToString());
285                 sb.AppendLine(selectedLevel);
286                 sb.AppendLine(durationRecorded.ToString());
287                 sb.Append("Usage rate: ").AppendLine(usageRate.ToString
                      ("#.##"));
288                 sb.Append("Duration fraction of hour: ").AppendLine
                      (durationFractionOfHour.ToString("#.##"));
289                 sb.Append("Used ").AppendLine(usedInActivity.ToString
                      ("#.##"));
290
291                 MessageBox.Show(sb.ToString());
292
293                 //-- Return the Activity object constructed from the values
294                 //-- Harvested from the form.
295                 return tempActivity;
296             }
297         }
298         #endregion
299
300         #region Display string methods
301         //-- string with a single activity
302         private string MakeSingleDisplayString(Activity activity)
303         {
304             var tempString = string.Empty;
305             var sb = new StringBuilder();
306             var usedString = Convert.ToInt32(activity.Used).ToString();
307
308             if(activity.MachineName.Length >= 12)
309             {
310                 sb.Append(activity.MachineName).Append("\t").Append
                      (activity.Duration.ToString()).Append("\t").AppendLine
```

```
                    (usedString);
            }
            else //-- Need an extra tab for alignment
            {
                sb.Append(activity.MachineName).Append("\t").Append
                  ("\t").Append(activity.Duration.ToString()).Append
                  ("\t").AppendLine(usedString);
            }


            return sb.ToString();
        }

        //-- Whole list of activities as strings in single string
        private string MakeWholeDisplayString(List<string> displayList)
        {
            var sb = new StringBuilder();
            sb.Append("Machine").Append("\t").Append("\t").Append
              ("Minutes").Append("\t").AppendLine("Used");

            foreach (var line in displayList)
            {
                sb.Append(line);
            }

            return sb.ToString();
        }
        #endregion



        #region Data methods for population of Combos
        /// <summary>
        /// Get the list of machines from the text file
        /// </summary>
        /// <returns></returns>
        private List<string> Machines()
        {
            return Lists.Machines();
        }

        /// <summary>
        ///
        /// </summary>
        /// <returns></returns>
        private List<string> Levels()
        {
            return Lists.Levels();
        }

        private double FractionOfHour(int minutes)
        {
            //-- in order to return double
            //-- the int minutes must be cast to double
            //-- before division
            return (double)minutes / 60;
```

```
363             }
364
365             #endregion
366
367
368         }
369     }
370
```