

```
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 default unless Treadmill      ↗
64      selected
65      InclinedCheckBoxLabel.Visibility = Visibility.Hidden;
66      InclinedCheckBox.Visibility = Visibility.Hidden;
67  }
68
69  //-- alternative constructor for when loaded without a person object ↗
70  available
71  public PageActivity()
72  {
73      InitializeComponent();
74
75      //-- Hide the inclined controls by default unless Treadmill      ↗
76      selected
77      InclinedCheckBoxLabel.Visibility = Visibility.Hidden;
78      InclinedCheckBox.Visibility = Visibility.Hidden;
79
80      //-- set up list headers
81      var headers = "Machine" + "\t" + "Used" + "\t" + "used";
82      //-- add to list
83      activitiesToDisplay.Add(headers);
84  }
85  #endregion
86
87  #region 04 Loaded
88  //-- Executes on page loaded
89  private void Page_Loaded(object sender, RoutedEventArgs e)
90  {
91      //-- confirm the person data has been received
92      MessageBox.Show("Person data received for " + person.PersonName);
93  }
94  #endregion
95
96  #region Navigation Buttons
97  private void BackButton_Click(object sender, RoutedEventArgs e)
98  {
99      var pagePerson = new PagePerson();
100      this.NavigationService.Navigate(pagePerson);
101  }
102
103  private void PageSummaryButton_Click(object sender, RoutedEventArgs e)
104  {
105      //-- finalise the summary object
106      summary.NumberOfActivities = recordedActivities.Count;
107      summary.MinutesOfExercise = totalMinutesOfExercise;
108      summary.TotalUsed = (int)usedRunningTotal;
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,           ↗
129         SelectionChangedEventArgs e)
130     {
131         var selectedMachineCombo = sender as ComboBox;
132         selectedMachine = selectedMachineCombo.SelectedItem as string;
133
134         //-- Control the visibility of the checkbox control label and    ↗
135         checkbox
136         if (selectedMachine == "Treadmill")
137         {
138             InclinedCheckBoxLabel.Visibility = Visibility.Visible;
139             InclinedCheckBox.Visibility = Visibility.Visible;
140         }
141         else
142         {
143             InclinedCheckBoxLabel.Visibility = Visibility.Hidden;
144             InclinedCheckBox.Visibility = Visibility.Hidden;
145         }
146     }
147     #endregion
148
149     #region Levels Combo handler methods
150     private void LevelsCombo_Loaded(object sender, RoutedEventArgs e)
151     {
152         var combo = sender as ComboBox;
153         combo.ItemsSource = Levels();
154         combo.SelectedIndex = 0;
155     }
156
157     private void LevelsCombo_SelectionChanged(object sender,           ↗
158         SelectionChangedEventArgs e)
159     {
160         var selectedLevelCombo = sender as ComboBox;
161         selectedLevel = selectedLevelCombo.SelectedItem as string;
162
163         ////-- DEBUG
164         //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 object 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 process 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
196             //-- Reset text box
197             DurationTextBox.Text = "";
198
199             //-- Manage display string, first this activity
200             //-- to be added to list for display
201             var listString = MakeSingleDisplayString(currentActivity);
202             activitiesToDisplay.Add(listString);
203
204             //-- refresh and display list of activities
205             //-- by making one string from the whole list
206             ActivityListTextBlock.Text = MakeWholeDisplayString
207             (activitiesToDisplay);
208         }
209         else // Message that limit exceeded
210         {
211             MessageBox.Show("The maximum number of activities is 6");
212         }
213     }
214
215     private Activity HarvestForm
216     {
217         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
232                 (DurationTextBox.Text);
233             //-- check it meets length rule
234             if (ActivityValidation.ActivityDurationValid
235                 (durationRecordedToCheck))
236             {
237                 durationRecorded = Convert.ToInt32
238                     (DurationTextBox.Text);
239             }
240             else //-- Outcome for value outside the rule 5 - 60
241                 minutes
242             {
243                 MessageBox.Show(" The activity duration must be
244                     between 5 and 60 mminutes");
245             }
246         }
247         else //-- Outcome for empty duration text box
248         {
249             MessageBox.Show("You must enter a duration for the
250                 activity");
251         }
252
253         //-- convert minutes to a fraction of an hour.
254         durationFractionOfHour = FractionOfHour(durationRecorded);
255
256         //-- Get the Rate for combination of Machine and Level
257         usageRate = MachineDataDb.GetRate(selectedMachine,
258             selectedLevel);
259
260         //-- If the weighted/inlined check box true, usage is
261             increased by 11%
262         //-- Else, straightforward multiplication of Rate per Hour *
263             Fraction of hour recorded
264         if (isWeighted)
265         {
266             usedInActivity = (usageRate * durationFractionOfHour) *
267                 weightingFactor;
268         }
269         else
270         {
271             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);
311     }
312     else //-- Need an extra tab for alignment
313     {
314         sb.Append(activity.MachineName).Append("\t").Append
            (" \t").Append(activity.Duration.ToString()).Append
            (" \t").AppendLine(usedString);
315     }
316
317
318     return sb.ToString();
319 }
320
321 //-- Whole list of activities as strings in single string
322 private string MakeWholeDisplayString(List<string> displayList)
323 {
324     var sb = new StringBuilder();
325     sb.Append("Machine").Append("\t").Append("\t").Append
            ("Minutes").Append("\t").AppendLine("Used");
326
327     foreach (var line in displayList)
328     {
329         sb.Append(line);
330     }
331
332     return sb.ToString();
333 }
334 #endregion
335
336
337
338 #region Data methods for population of Combos
339 /// <summary>
340 /// Get the list of machines from the text file
341 /// </summary>
342 /// <returns></returns>
343 private List<string> Machines()
344 {
345     return Lists.Machines();
346 }
347
348 /// <summary>
349 ///
350 /// </summary>
351 /// <returns></returns>
352 private List<string> Levels()
353 {
354     return Lists.Levels();
355 }
356
357 private double FractionOfHour(int minutes)
358 {
359     //-- in order to return double
360     //-- the int minutes must be cast to double
361     //-- before division
362     return (double)minutes / 60;
```

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