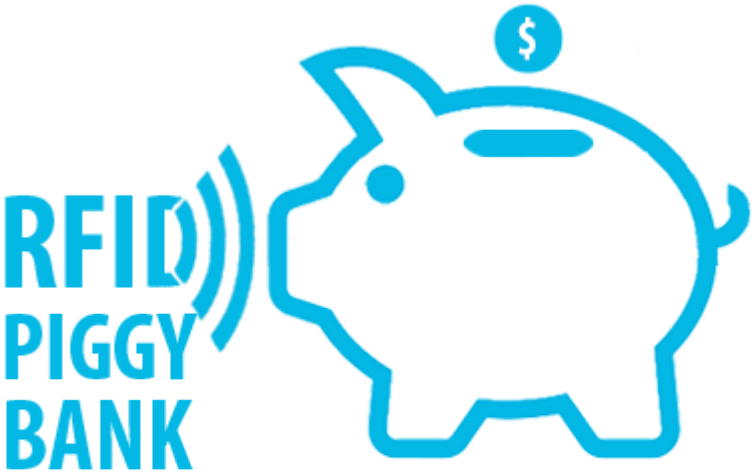
TPI 2018 - RFID Piggy Bank



Jean-Daniel Küenzi
I.DA-P4A
École d'informatique (CFPT-I)

4 Mai 2018

Table des matières

Chapitre 1

RFID Piggy Bank

1.1 Vue

1.1.1 Program.cs

```
1  /*
2   * Author    : Küenzi Jean-Daniel
3   * Date      : 09.05.2018
4   * Desc.     : This is the main program, where logic and calls to other ↩
       classes are made
5   * Version   : 1.0.0
6   */
7  using System;
8  using System.Collections;
9  using System.Threading;
10
11 using Microsoft.SPOT;
12 using Microsoft.SPOT.Presentation;
13 using Microsoft.SPOT.Presentation.Controls;
14 using Microsoft.SPOT.Presentation.Media;
15 using Microsoft.SPOT.Presentation.Shapes;
16 using Microsoft.SPOT.Touch;
17
18 using GHI.Pins;
19 using Gadgeteer.Networking;
20 using GT = Gadgeteer;
21 using GTM = Gadgeteer.Modules;
22 using Gadgeteer.Modules.GHIElectronics;
23 using Microsoft.SPOT.Hardware;
24
25 namespace RFIDPiggyBank
26 {
27     public partial class Program
28     {
29         /// <summary>
30         /// The state of the menu
31         /// </summary>
32         private enum MENU_STATE { initial, addCard, deleteCard, ↩
              displayCards, secretCode };
33
34         /// <summary>
35         /// The state of the servomotor, we us open and close because ↩
              lock is a reserved type (open = unlock & close = lock)
36         /// </summary>
37         private enum SERVO_STATE { open, close };
38
39         /// <summary>
40         /// The secret sequel to the code (it looks like konami code)
41         /// </summary>
42         private enum SECRET_CODE { up1, up2, down1, down2, left1, ↩
              right1, left2, right2, success, error };
43
44         /// <summary>
```

```
45          /// The state sequel when a card (RFID badge) is scan
46          /// </summary>
47          private enum SCAN_CARD_STATE { waitRFID, RFIDDetected, ↩
                RFIDValid, RFIDInvalid };
48
49          /// <summary>
50          /// The state sequel when we add a card (RFID Badge)
51          /// </summary>
52          private enum ADD_CARD_STATE { waitRFID, RFIDDetected, ↩
                badgeExist, bageDontExist, save, errorMSG, successMSG };
53
54          /// <summary>
55          /// The state sequel when we display all cards (RFID Badges)
56          /// </summary>
57          private enum DISPLAY_CARDS_STATE { listIsEmpty, errorMSG, ↩
                displayAllCards };
58
59          /// <summary>
60          /// The state sequel when we delete a card (RFID Badge)
61          /// </summary>
62          private enum DELETE_CARD_STATE { listIsEmpty, selectCard, save, ↩
                errorMSG, success };
63
64          /// <summary>
65          /// Constant for the menu position on the LCD
66          /// </summary>
67          private const int MENU1_Y = 10;
68          private const int MENU2_Y = 30;
69          private const int MENU3_Y = 50;
70          private const int MENU4_Y = 70;
71
72          /// <summary>
73          /// Constant of the value when the joystick is up or right /!\ ↩
                It's can change with the orientation of the josytick /!\
74          /// </summary>
75          private const double JOYSTICK_UP_RIGHT = 0.4;
76
77          /// <summary>
78          /// Constant of the value if the joystick is down ot left /!\ ↩
                It's can change with the orientation of the josytick /!\
79          /// </summary>
80          private const double JOYSTICK_DOWN_LEFT = 0.6;
81
82          /// <summary>
83          /// This is the logic version of the main menu
84          /// </summary>
85          private int _menu = 0;
86
87          private MENU_STATE _menuState = MENU_STATE.initial;
88          private SERVO_STATE _servoState = SERVO_STATE.close;
89          private SECRET_CODE _secretState = SECRET_CODE.up1;
90          private SCAN_CARD_STATE _scanCardState = SCAN_CARD_STATE.waitRFID;
91          private ADD_CARD_STATE _addCardState = ADD_CARD_STATE.waitRFID;
92          private DISPLAY_CARDS_STATE _displayCardsState = ↩
                DISPLAY_CARDS_STATE.listIsEmpty;
93          private DELETE_CARD_STATE _deleteCardState = ↩
                DELETE_CARD_STATE.listIsEmpty;
94
95          /// <summary>
96          /// 90000ms = 1min30 -> this is the secure timer if we have ↩
                forgotten to close the box
97          /// </summary>
98          private GT.Timer _secuTimer = new GT.Timer(90000);
99
100         /// <summary>
101         /// The axe X of the joystick
102         /// </summary>
103         private AnalogInput _joystickX = new ↩
                AnalogInput(FEZSpider.Socket9.AnalogInput4);
104
105         /// <summary>
106         /// The axe Y of the joystick
```

```
107         /// </summary>
108         private AnalogInput _joystickY = new ↩
                AnalogInput(FEZSpider.Socket9.AnalogInput5);
109
110         /// <summary>
111         /// The joystick button
112         /// </summary>
113         private InterruptPort _joystickButton = new ↩
                InterruptPort(FEZSpider.Socket9.Pin3, false, ↩
                Port.ResistorMode.Disabled, ↩
                Port.InterruptMode.InterruptEdgeHigh);
114
115         /// <summary>
116         /// This method is run when the mainboard is powered up or reset.
117         /// </summary>
118         void ProgramStarted()
119         {
120             InitializesClassesData();
121
122             GT.Timer timer = new GT.Timer(100);// Perdiod = every 0.1 ↩
                    seconds (100ms)
123             timer.Tick += timer_Tick;           // We do a timer because ↩
                    with Gageteer (GHI) while(true) blocks the thread
124             timer.Start();                      // and the acces to the ↩
                    components
125
126             _joystickButton.OnInterrupt += _joystickButton_OnInterrupt;
127
128             _secuTimer.Tick += _secuTimer_Tick;
129         }
130
131         private void _secuTimer_Tick(GT.Timer pbTimer)
132         {
133             _servoState = SERVO_STATE.close;
134         }
135
136         /// <summary>
137         /// This timer event serves as the main sequencer of the program
138         /// </summary>
139         /// <param name="pbTimer">The GT.Timer who's calling (GT = ↩
                Gadgeteer)</param>
140         private void timer_Tick(GT.Timer pbTimer)
141         {
142             // Main menu
143             if (_menuState == MENU_STATE.initial)
144             {
145                 InitialState();
146             }
147
148             // Add card event
149             if (_menuState == MENU_STATE.addCard)
150             {
151                 AddCard();
152             }
153
154             // Delete card event
155             if (_menuState == MENU_STATE.deleteCard)
156             {
157                 DeleteCard();
158             }
159
160             // Display cards event
161             if (_menuState == MENU_STATE.displayCards)
162             {
163                 DisplayCards();
164             }
165
166             // Unlock with the secret code event
167             if (_menuState == MENU_STATE.secretCode)
168             {
169                 UnlockSecretCode();
170             }
```

```
171         }
172
173         /// <summary>
174         /// This method initializes the different classes and data
175         /// </summary>
176         private void InitializesClassesData()
177         {
178             ServoMotor.GetInstance().Lock();
179             RFIDReader.GetInstance();
180             DisplayLoad();
181             ListOfCards.GetInstance().CardsList = ↩
                    SDCard.GetInstance().LoadCards();
182             if (ListOfCards.GetInstance().CardsList == null)
183             {
184                 ListOfCards.GetInstance().CardsList = new ArrayList();
185             }
186             RestoreInitialState();
187         }
188
189         /// <summary>
190         /// This method interrupt all threads when the joystick is press
191         /// </summary>
192         /// <param name="pbData1"></param>
193         /// <param name="pbData2"></param>
194         /// <param name="pbTime"></param>
195         private void _joystickButton_OnInterrupt(uint pbData1, uint ↩
                pbData2, DateTime pbTime)
196         {
197             // We do this only if we are on the main menu
198             if (_menuState == MENU_STATE.initial)
199             {
200                 switch (_menu)
201                 {
202                     case 0:
203                         _menuState = MENU_STATE.initial;
204                         break;
205                     case 1:
206                         _menuState = MENU_STATE.addCard;
207                         break;
208                     case 2:
209                         _menuState = MENU_STATE.deleteCard;
210                         break;
211                     case 3:
212                         _menuState = MENU_STATE.displayCards;
213                         break;
214                     case 4:
215                         _menuState = MENU_STATE.secretCode;
216                         break;
217                     default:
218                         _menuState = MENU_STATE.initial;
219                         break;
220                 }
221                 DeleteCurrentBadgescan();
222                 LCD.GetInstance().Clear();
223             }
224         }
225
226         /// <summary>
227         /// This method clear the current badge that is scan by the ↩
                RFIDReader
228         /// </summary>
229         private void DeleteCurrentBadgescan()
230         {
231             RFIDReader.GetInstance().CurrentUid = "";
232             RFIDReader.GetInstance().IsBadgeScan = false;
233         }
234
235         /// <summary>
236         /// This method restore to initial state (main menu)
237         /// </summary>
238         private void RestoreInitialState()
239         {
```

```
240             _menu = 0;
241             _menuState = MENU_STATE.initial;
242             _servoState = SERVO_STATE.close;
243             _secretState = SECRET_CODE.up1;
244             _scanCardState = SCAN_CARD_STATE.waitRFID;
245             _addCardState = ADD_CARD_STATE.waitRFID;
246             _displayCardsState = DISPLAY_CARDS_STATE.listIsEmpty;
247             _deleteCardState = DELETE_CARD_STATE.listIsEmpty;
248 
249             // Refresh the LCD text fields
250             LCDTextFields.Content = Card.DEFAULT_NAME;
251             LCDTextFields.CursorPosition = 0;
252             LCDTextFields.ShouldBeRefresh = true;
253 
254             DeleteCurrentBadgescan();
255             LCD.GetInstance().Clear();
256             DisplayMainMenu(_menu);
257         }
258 
259         /// <summary>
260         /// This method display an error message on the lcd after ↩
                clearing it
261         /// </summary>
262         private void DisplayError()
263         {
264             LCD.GetInstance().Clear();
265             LCD.GetInstance().DisplayText(GT.Color.Red, "/!\\ Une ↩
                    erreur est survenue /!\\", 10, ↩
                    LCD.GetInstance().LcdHeight - 20);
266         }
267 
268         /// <summary>
269         /// This method display an save message on the lcd after ↩
                clearing it
270         /// </summary>
271         private void DisplaySave()
272         {
273             LCD.GetInstance().Clear();
274             LCD.GetInstance().DisplayText(GT.Color.Gray, "Sauvegarde en ↩
                    cours...", 10, LCD.GetInstance().LcdHeight - 20);
275         }
276 
277         /// <summary>
278         /// This method display an load message on the lcd after ↩
                clearing it
279         /// </summary>
280         private void DisplayLoad()
281         {
282             LCD.GetInstance().Clear();
283             LCD.GetInstance().DisplayText(GT.Color.Gray, "Chargement en ↩
                    cours...", 10, LCD.GetInstance().LcdHeight - 20);
284         }
285 
286         /// <summary>
287         /// This method display the main menu on the LCD
288         /// </summary>
289         /// <param name="pbMenu">The logic version of the main menu ↩
                (Index)</param>
290         private void DisplayMainMenu(int pbMenu)
291         {
292             LCD.GetInstance().DisplayText(Gadgeteer.Color.Black, ↩
                    "Ajouter un badge", 10, MENU1_Y);
293             LCD.GetInstance().DisplayText(Gadgeteer.Color.Black, ↩
                    "Supprimer un badge", 10, MENU2_Y);
294             LCD.GetInstance().DisplayText(Gadgeteer.Color.Black, ↩
                    "Afficher la liste des badges", 10, MENU3_Y);
295             LCD.GetInstance().DisplayText(Gadgeteer.Color.Black, ↩
                    "Deverouiller avec le code secret", 10, MENU4_Y);
296             switch (pbMenu)
297             {
298                 case 1:
```

```
299                     LCD.GetInstance().DisplayText(Gadgeteer.Color.Blue, ↩
                            "Ajouter un badge", 10, MENU1_Y);
300                     break;
301                 case 2:
302                     LCD.GetInstance().DisplayText(Gadgeteer.Color.Blue, ↩
                            "Supprimer un badge", 10, MENU2_Y);
303                     break;
304                 case 3:
305                     LCD.GetInstance().DisplayText(Gadgeteer.Color.Blue, ↩
                            "Afficher la liste des badges", 10, MENU3_Y);
306                     break;
307                 case 4:
308                     LCD.GetInstance().DisplayText(Gadgeteer.Color.Blue, ↩
                            "Deverouiller avec le code secret", 10, MENU4_Y);
309                     break;
310             }
311         }
312 
313         /// <summary>
314         /// When no menu is selected
315         /// </summary>
316         private void InitialState()
317         {
318             switch (_scanCardState)
319             {
320                 case SCAN_CARD_STATE.waitRFID:
321                     if (RFIDReader.GetInstance().IsBadgeScan)
322                     {
323                         _scanCardState = SCAN_CARD_STATE.RFIDDetected;
324                     }
325                     break;
326                 case SCAN_CARD_STATE.RFIDDetected:
327                     if (_servoState == SERVO_STATE.close) // If the ↩
                            servo is lock
328                     {
329                         bool isValid = ↩
                                ListOfCards.GetInstance().FindCardInlist(RFIDReader.GetIn
330                         if (isValid)
331                         {
332                             _scanCardState = SCAN_CARD_STATE.RFIDValid;
333                         }
334                         else
335                         {
336                             _scanCardState = SCAN_CARD_STATE.RFIDInvalid;
337                         }
338                     }
339                     else
340                     {
341                         _scanCardState = SCAN_CARD_STATE.RFIDInvalid;
342                     }
343                     break;
344                 case SCAN_CARD_STATE.RFIDValid:
345                     _servoState = SERVO_STATE.open;
346                     _scanCardState = SCAN_CARD_STATE.waitRFID;
347                     DeleteCurrentBadgescan();
348                     break;
349                 case SCAN_CARD_STATE.RFIDInvalid:
350                     _servoState = SERVO_STATE.close;
351                     _scanCardState = SCAN_CARD_STATE.waitRFID;
352                     DeleteCurrentBadgescan();
353                     break;
354                 default:
355                     _scanCardState = SCAN_CARD_STATE.waitRFID;
356                     break;
357             }
358 
359             switch (_servoState)
360             {
361                 case SERVO_STATE.open:
362                     ServoMotor.GetInstance().Unlock();
363                     _secuTimer.Start();
364                     break;
```

```
365                 case SERVO_STATE.close:
366                     ServoMotor.GetInstance().Lock();
367                     _secuTimer.Stop();
368                     break;
369                 default:
370                     _servoState = SERVO_STATE.close;
371                     break;
372             }
373 
374             if (_joystickX.Read() > JOYSTICK_DOWN_LEFT) // If the joystick is down
375             {
376                 _menu++;
377                 if (_menu > 4)
378                 {
379                     _menu = 0;
380                 }
381                 DisplayMainMenu(_menu); // It is not removed from the test to prevent the lcd flashing
382                 Thread.Sleep(100); // Wait 0.1 second to prevent the menu from scrolling too fast
383             }
384             else if (_joystickX.Read() < JOYSTICK_UP_RIGHT) // The joystick is up
385             {
386                 _menu--;
387                 if (_menu < 0)
388                 {
389                     _menu = 4;
390                 }
391                 DisplayMainMenu(_menu); // It is not removed from the test to prevent the lcd flashing
392                 Thread.Sleep(100); // Wait 0.1 second to prevent the menu from scrolling too fast
393             }
394         }
395 
396         /// <summary>
397         /// When the menu to add a card is selected
398         /// </summary>
399         private void AddCard()
400         {
401             switch (_addCardState)
402             {
403                 case ADD_CARD_STATE.waitRFID:
404                     LCD.GetInstance().Clear();
405                     LCD.GetInstance().DisplayText(GT.Color.Gray, "Veuillez approcher un badge du lecteur");
406                     if (RFIDReader.GetInstance().IsBadgeScan)
407                     {
408                         _addCardState = ADD_CARD_STATE.RFIDDetected;
409                         LCD.GetInstance().DisplayText(GT.Color.Green, "Votre badge a ete correctement scanne", 10, LCD.GetInstance().LcdHeight / 2);
410                         Thread.Sleep(2000); // Wait 2 seconds to see the message
411                     }
412                     break;
413                 case ADD_CARD_STATE.RFIDDetected:
414                     LCD.GetInstance().Clear();
415                     string uid = RFIDReader.GetInstance().CurrentUid;
416 
417                     if (ListOfCards.GetInstance().FindCardInlist(uid)) // If the badge scanned already exist
418                     {
419                         _addCardState = ADD_CARD_STATE.badgeExist;
420                     }
421                     else
422                     {
423                         _addCardState = ADD_CARD_STATE.bageDontExist;
424                     }
425                     break;
```

```
426                 case ADD_CARD_STATE.badgeExist:
427                     LCD.GetInstance().DisplayText(GT.Color.Red, "/!\\ Erreur : Ce badge est deja sauvegarde /!\\", 10, LCD.GetInstance().LcdHeight / 2);
428                     Thread.Sleep(2000); // Wait 2 seconds to see the message
429                     RestoreInitialState();
430                     break;
431                 case ADD_CARD_STATE.bageDontExist:
432                     string name = LCDTextFields.Content; // The content value of the LCD field, it's the name of the badge
433                     char[] charArray = name.ToCharArray(); // We split the name in a char array to make it easier to modify char by char
434                     int x = 110; // The position index where we're gonna write the first char
435 
436                     if (LCDTextFields.ShouldBeRefresh) // If we need to refresh because we have modify a char or the position of the cursor
437                     {
438                         LCD.GetInstance().Clear();
439                         LCD.GetInstance().DisplayText(GT.Color.Gray, "Votre badge :", 10, LCD.GetInstance().LcdHeight / 2);
440                         LCD.GetInstance().DisplayText(GT.Color.Gray, "Pour valider le nom, appuyer sur le joystick", 10, LCD.GetInstance().LcdHeight - 20);
441                         for (int i = 0; i < charArray.Length; i++)
442                         {
443                             if (i == LCDTextFields.CursorPosition) // If the cursorposition is at this char
444                                 LCD.GetInstance().DisplayText(GT.Color.Blue, charArray[i].ToString(), x, LCD.GetInstance().LcdHeight / 2);
445                             else
446                                 LCD.GetInstance().DisplayText(GT.Color.Black, charArray[i].ToString(), x, LCD.GetInstance().LcdHeight / 2);
447                             x += 10; // Increment the X position on the LCD
448                         }
449                         LCDTextFields.ShouldBeRefresh = false;
450                     }
451 
452                     if (_joystickX.Read() < JOYSTICK_UP_RIGHT) // If the joystick is up
453                     {
454                         charArray[LCDTextFields.CursorPosition]++; // Increment the char ex : A -> B
455                         LCDTextFields.ShouldBeRefresh = true;
456                         Thread.Sleep(100); // Wait 0.1 second to prevent the letter from scrolling too fast
457                     }
458                     else if (_joystickX.Read() > JOYSTICK_DOWN_LEFT) // If the joystick is down
459                     {
460                         charArray[LCDTextFields.CursorPosition]--; // ecrement the char ex : B -> A
461                         LCDTextFields.ShouldBeRefresh = true;
462                         Thread.Sleep(100); // Wait 0.1 second to prevent the letter from scrolling too fast
463                     }
464 
465 
466                     if (_joystickY.Read() < JOYSTICK_UP_RIGHT) // If the joystick is right
467                     {
468                         LCDTextFields.CursorPosition++; // Move the cursor to the next char
```

```
469                         if (LCDTextFields.CursorPosition > ↩
                                charArray.Length - 1) // If the cursor get ↩
                                out of the range of the char array
470                         {
471                             LCDTextFields.CursorPosition = 0; // Move ↩
                                    to the first position of the char array
472                         }
473                         LCDTextFields.ShouldBeRefresh = true;
474                         Thread.Sleep(200); // Wait 0.2 seconds to ↩
                                prevent the cursor from moving too fast
475                     }
476                     else if (_joystickY.Read() > JOYSTICK_DOWN_LEFT) // ↩
                            If the joystick is left
477                     {
478                         LCDTextFields.CursorPosition--; // Move the ↩
                                cursor to the previous char
479                         if (LCDTextFields.CursorPosition < 0) // If the ↩
                                cursor get out of the range of the char array
480                         {
481                             LCDTextFields.CursorPosition = ↩
                                    charArray.Length - 1; // Move to the ↩
                                    last position of the char array
482                         }
483                         LCDTextFields.ShouldBeRefresh = true;
484                         Thread.Sleep(200); // Wait 0.2 seconds to ↩
                                prevent the cursor from moving too fast
485                     }
486
487                     LCDTextFields.Content = new string(charArray); // ↩
                            Set the LCD text field with the value of the ↩
                            modify char array
488
489                     if (!_joystickButton.Read()) // If joystick button ↩
                            is press
490                     {
491                         _addCardState = ADD_CARD_STATE.save;
492                     }
493                     break;
494                 case ADD_CARD_STATE.save:
495                     try
496                     {
497                         uid = RFIDReader.GetInstance().CurrentUid; // ↩
                                Get the uid of the badge that was scanned
498                         name = LCDTextFields.Content;
499                         ListOfCards.GetInstance().AddCardToList(name, ↩
                                uid);
500                         SDCard.GetInstance().SaveCards(ListOfCards.GetInstance().Car
501                         _addCardState = ADD_CARD_STATE.successMSG;
502                     }
503                     catch (Exception e)
504                     {
505                         _addCardState = ADD_CARD_STATE.errorMSG;
506                     }
507                     break;
508                 case ADD_CARD_STATE.errorMSG:
509                     DisplayError();
510                     Thread.Sleep(2000);
511                     RestoreInitialState();
512                     break;
513                 case ADD_CARD_STATE.successMSG:
514                     DisplaySave();
515                     LCD.GetInstance().DisplayText(GT.Color.Green, "Le ↩
                            badge a bien ete ajoute", 10, ↩
                            LCD.GetInstance().LcdHeight / 2);
516                     Thread.Sleep(2000);
517                     RestoreInitialState();
518                     break;
519                 default:
520                     _addCardState = ADD_CARD_STATE.waitRFID;
521                     break;
522             }
523         }
```

```
524
525         /// <summary>
526         /// When the menu to delete a card is selected
527         /// </summary>
528         private void DeleteCard()
529         {
530 
531             switch (_deleteCardState)
532             {
533                 case DELETE_CARD_STATE.listIsEmpty:
534                     LCD.GetInstance().Clear();
535                     if (ListOfCards.GetInstance().IsEmpty())
536                     {
537                         LCD.GetInstance().DisplayText(GT.Color.Red, ↩
                            "/!\\ Aucun badge n'est enregistre /!\\", ↩
                            10, LCD.GetInstance().LcdHeight / 2);
538                         Thread.Sleep(2000);
539                         RestoreInitialState();
540                     }
541                     else
542                     {
543                         _deleteCardState = DELETE_CARD_STATE.selectCard;
544                     }
545                     break;
546                 case DELETE_CARD_STATE.selectCard:
547                     int positionY = 10; // The Y position on the LCD
548                     int i = 0;
549                     foreach (Card card in ↩
                        ListOfCards.GetInstance().CardsList)
550                     {
551                         if (LCDTextFields.CursorPosition == i) // If ↩
                            the cursorposition is at this char
552                             LCD.GetInstance().DisplayText(GT.Color.Blue, ↩
                                card.Name, 10, positionY);
553                         else
554                             LCD.GetInstance().DisplayText(GT.Color.Black, ↩
                                card.Name, 10, positionY);
555                         positionY += 15; // Increment the Y position
556                         i++;
557                     }
558                     LCD.GetInstance().DisplayText(GT.Color.Gray, "Pour ↩
                        selectionner le badge, appuyer sur le joystick", ↩
                        10, LCD.GetInstance().LcdHeight - 30);
559 
560                     if (_joystickX.Read() > JOYSTICK_DOWN_LEFT) // If ↩
                        joystick is down
561                     {
562                         LCDTextFields.CursorPosition++; // Move the ↩
                            cursor to the next name
563                         if (LCDTextFields.CursorPosition > ↩
                            ListOfCards.GetInstance().CardsList.Count - ↩
                            1) // If the cursor get out of the range of ↩
                            the list of cards array
564                         {
565                             LCDTextFields.CursorPosition = 0; // Move ↩
                                to the first position of the list of ↩
                                cards array
566                         }
567                         Thread.Sleep(100); // Wait 0.1 second to ↩
                            prevent the cursor from moving too fast
568                     }
569                     else if (_joystickX.Read() < JOYSTICK_UP_RIGHT) // ↩
                        If joystick is up
570                     {
571                         LCDTextFields.CursorPosition--; // Move the ↩
                            cursor to the previous name
572                         if (LCDTextFields.CursorPosition < 0) // If the ↩
                            cursor get out of the range of the list array
573                         {
574                             LCDTextFields.CursorPosition = ↩
                                ListOfCards.GetInstance().CardsList.Count ↩
                                - 1; // Move to the last position of the ↩
```

```
                                list of cards array
575                         }
576                         Thread.Sleep(100); // Wait 0.1 second to ↩
                               prevent the cursor from moving too fast
577                     }
578                     if (!_joystickButton.Read()) // If joystick button ↩
                           is press
579                     {
580                         _deleteCardState = DELETE_CARD_STATE.save;
581                     }
582                     break;
583                 case DELETE_CARD_STATE.save:
584                     try
585                     {
586                         ListOfCards.GetInstance().DeleteCardFromList(LCDTextFields.C
587                         SDCard.GetInstance().SaveCards(ListOfCards.GetInstance().Car
588                         _deleteCardState = DELETE_CARD_STATE.success;
589                     }
590                     catch (Exception e)
591                     {
592                         _deleteCardState = DELETE_CARD_STATE.errorMSG;
593                     }
594                     break;
595                 case DELETE_CARD_STATE.errorMSG:
596                     DisplayError();
597                     Thread.Sleep(2000);
598                     RestoreInitialState();
599                     break;
600                 case DELETE_CARD_STATE.success:
601                     DisplaySave();
602                     LCD.GetInstance().DisplayText(GT.Color.Green, "Le ↩
                           badge a bien ete supprime", 10, ↩
                           LCD.GetInstance().LcdHeight / 2);
603                     Thread.Sleep(2000);
604                     RestoreInitialState();
605                     break;
606                 default:
607                     _deleteCardState = DELETE_CARD_STATE.listIsEmpty;
608                     break;
609             }
610         }
611
612         /// <summary>
613         /// When the menu to display all cards is selected
614         /// </summary>
615         private void DisplayCards()
616         {
617             switch (_displayCardsState)
618             {
619                 case DISPLAY_CARDS_STATE.listIsEmpty:
620                     LCD.GetInstance().Clear();
621                     if (ListOfCards.GetInstance().IsEmpty())
622                     {
623                         _displayCardsState = DISPLAY_CARDS_STATE.errorMSG;
624                     }
625                     else
626                     {
627                         _displayCardsState = ↩
                               DISPLAY_CARDS_STATE.displayAllCards;
628                     }
629                     break;
630                 case DISPLAY_CARDS_STATE.errorMSG:
631                     LCD.GetInstance().DisplayText(GT.Color.Red, "/!\\ ↩
                           Aucun badge n'est enregistre /!\\", 10, ↩
                           LCD.GetInstance().LcdHeight / 2);
632                     Thread.Sleep(2000);
633                     RestoreInitialState();
634                     break;
635                 case DISPLAY_CARDS_STATE.displayAllCards:
636                     int positionY = 10; // The Y position on the LCD
637                     LCD.GetInstance().DisplayText(GT.Color.Gray, "Pour ↩
                           quitter, appuyer sur le joystick", 10, ↩
```

```
                    LCD.GetInstance().LcdHeight - 20);
638
639                 foreach (Card card in ↩
                        ListOfCards.GetInstance().CardsList)
640                 {
641                     LCD.GetInstance().DisplayText(GT.Color.Black, ↩
                            card.Name, 10, positionY);
642                     positionY += 15; // Increment the Y position
643                 }
644
645                 if (!_joystickButton.Read()) // If joystick button ↩
                        is press
646                 {
647                     RestoreInitialState();
648                 }
649                 break;
650             default:
651                 _displayCardsState = DISPLAY_CARDS_STATE.listIsEmpty;
652                 break;
653         }
654     }
655
656     /// <summary>
657     /// When the menu to unlock the box with the secret code is ↩
            selected
658     /// </summary>
659     private void UnlockSecretCode()
660     {
661         int nbrClue = 0; // This is for the number of * that we're ↩
                going to wrote on the LCD
662         int positionX = 10; // Position X ont the LCD
663
664         LCD.GetInstance().DisplayText(GT.Color.Black, "Progression ↩
                :", positionX, LCD.GetInstance().LcdHeight / 2);
665
666         LCD.GetInstance().DisplayText(GT.Color.Gray, "Pour quitter, ↩
                appuyer sur le joystick", positionX, ↩
                LCD.GetInstance().LcdHeight - 20);
667         bool oldJoystickread = ((_joystickX.Read() > ↩
                JOYSTICK_UP_RIGHT && _joystickX.Read() < ↩
                JOYSTICK_DOWN_LEFT) && // True if the joystick is int ↩
                the center
668                                 (_joystickY.Read() > ↩
                                        JOYSTICK_UP_RIGHT && ↩
                                        _joystickY.Read() < ↩
                                        JOYSTICK_DOWN_LEFT));
669
670         Thread.Sleep(200); // Wait 0.2 seconds to allow time to ↩
                move the joystick
671         bool joystickRead = ((_joystickX.Read() < JOYSTICK_UP_RIGHT ↩
                || _joystickX.Read() > JOYSTICK_DOWN_LEFT) || // True if ↩
                the joystick isn't at the center
672                              (_joystickY.Read() < JOYSTICK_UP_RIGHT ↩
                                    || _joystickY.Read() > ↩
                                    JOYSTICK_DOWN_LEFT));
673
674         if ((oldJoystickread && joystickRead) || _secretState == ↩
                SECRET_CODE.success || _secretState == ↩
                SECRET_CODE.error) // If the joystick was in the center ↩
                and then move
675         { ↩

            // or if the code is success || error
676             LCD.GetInstance().Clear();
677             switch (_secretState)
678             {
679                 case SECRET_CODE.up1:
680                     if (_joystickX.Read() <= JOYSTICK_UP_RIGHT) // ↩
                            If joystick is up
681                     {
682                         _secretState = SECRET_CODE.up2;
683                         nbrClue = 1;
```

```
684                             Debug.Print("1");
685                         }
686                         else
687                         {
688                             _secretState = SECRET_CODE.error;
689                         }
690                         break;
691                     case SECRET_CODE.up2:
692                         if (_joystickX.Read() <= JOYSTICK_UP_RIGHT) // ↩
                                If joystick is up
693                         {
694                             _secretState = SECRET_CODE.down1;
695                             nbrClue = 2;
696                             Debug.Print("2");
697                         }
698                         else
699                         {
700                             _secretState = SECRET_CODE.error;
701                         }
702                         break;
703                     case SECRET_CODE.down1:
704                         if (_joystickX.Read() >= JOYSTICK_DOWN_LEFT) // ↩
                                If joystick is down
705                         {
706                             _secretState = SECRET_CODE.down2;
707                             nbrClue = 3;
708                             Debug.Print("3");
709                         }
710                         else
711                         {
712                             _secretState = SECRET_CODE.error;
713                         }
714                         break;
715                     case SECRET_CODE.down2:
716                         if (_joystickX.Read() >= JOYSTICK_DOWN_LEFT) // ↩
                                If joystick is down
717                         {
718                             _secretState = SECRET_CODE.left1;
719                             nbrClue = 4;
720                             Debug.Print("4");
721                         }
722                         else
723                         {
724                             _secretState = SECRET_CODE.error;
725                         }
726                         break;
727                     case SECRET_CODE.left1:
728                         if (_joystickY.Read() >= JOYSTICK_DOWN_LEFT) // ↩
                                If joystick is left
729                         {
730                             _secretState = SECRET_CODE.right1;
731                             nbrClue = 5;
732                             Debug.Print("5");
733                         }
734                         else
735                         {
736                             _secretState = SECRET_CODE.error;
737                         }
738                         break;
739                     case SECRET_CODE.right1:
740                         if (_joystickY.Read() <= JOYSTICK_UP_RIGHT) // ↩
                                If joystick is right
741                         {
742                             _secretState = SECRET_CODE.left2;
743                             nbrClue = 6;
744                             Debug.Print("6");
745                         }
746                         else
747                         {
748                             _secretState = SECRET_CODE.error;
749                             nbrClue = 1;
750                         }
```

```
751                         break;
752                     case SECRET_CODE.left2:
753                         if (_joystickY.Read() >= JOYSTICK_DOWN_LEFT) // ↩
                               If joystick is left
754                         {
755                             _secretState = SECRET_CODE.right2;
756                             nbrClue = 7;
757                             Debug.Print("7");
758                         }
759                         else
760                         {
761                             _secretState = SECRET_CODE.error;
762                         }
763                         break;
764                     case SECRET_CODE.right2:
765                         if (_joystickY.Read() <= JOYSTICK_UP_RIGHT) // ↩
                               If joystick is right
766                         {
767                             _secretState = SECRET_CODE.success;
768                             Debug.Print("8");
769                         }
770                         else
771                         {
772                             _secretState = SECRET_CODE.error;
773                         }
774                         break;
775                     case SECRET_CODE.success:
776                         RestoreInitialState();
777                         _servoState = SERVO_STATE.open;
778                         break;
779                     case SECRET_CODE.error:
780                         _secretState = SECRET_CODE.up1;
781                         LCD.GetInstance().DisplayText(GT.Color.Red, ↩
                               "Code faux, veuillez recommencer");
782                         Thread.Sleep(1000);
783                         break;
784                     default:
785                         _secretState = SECRET_CODE.up1;
786                         break;
787                 }
788 
789                 for (int i = 0; i < nbrClue; i++)
790                 {
791                     LCD.GetInstance().DisplayText(GT.Color.Black, "*", ↩
                           100 + positionX, LCD.GetInstance().LcdHeight / 2 ↩
                           + 5);
792                     positionX += 10;
793                 }
794             }
795 
796             if (!_joystickButton.Read()) // If the joystick button is ↩
                   press
797             {
798                 RestoreInitialState();
799             }
800         }
801     }
802 }
```

Listing 1.1 – ./RFIDPiggyBank/RFIDPiggyBank/Program.cs

1.2 Modèles

1.2.1 Card.cs

```
/*
 * Author    : Küenzi Jean-Daniel
 * Date      : 09.05.2018
 * Desc.     : L'objet Card => Badge RFID
 * Version   : 1.0.0
 */
using System;
using Microsoft.SPOT;

namespace RFIDPiggyBank
{
    [Serializable]
    public class Card
    {
        /// <summary>
        /// Default name of a badge
        /// </summary>
        public const string DEFAULT_NAME = "Badge";

    /// <summary>
        /// The name of a badge
        /// </summary>
        private string _name;

    /// <summary>
        /// The uid of a badge
        /// </summary>
        private string _uid;

        public Card(string pbName, string pbUid)
        {
            Name = pbName;
            Uid = pbUid;
        }

    /// <summary>
        /// Getter and Setter for the name
        /// </summary>
        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }

    /// <summary>
        /// Getter and Setter for the name
        /// </summary>
        public string Uid
        {
            get { return _uid; }
            set { _uid = value; }
        }
    }
}
```

Listing 1.2 – ./RFIDPiggyBank/RFIDPiggyBank/Card.cs

1.2.2 LCD.cs

```
/*
 * Author    : Küenzi Jean-Daniel
 * Date      : 09.05.2018
 * Desc.     : Class that handles the TE35 (LCD Module) from GHI
 * Version   : 1.0.0
 */
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Gadgeteer;

using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Media;
using Microsoft.SPOT.Touch;

using GTM = Gadgeteer.Modules;
using Gadgeteer.Networking;
using System.IO;
using System.Text;

namespace RFIDPiggyBank
{
    public class LCD
    {
        /// <summary>
        /// The instance of the class LCD
        /// </summary>
        private static LCD _instance;

        /// <summary>
        /// The width of the TE35 Screen
        /// </summary>
        private int _lcdWidth;

        /// <summary>
        /// The height of the TE35 Screen
        /// </summary>
        private int _lcdHeight;
        /// <summary>
        /// The Display TE35 module using sockets 14, 13, 12 and 10 of the mainboard.
        /// </summary>
        private Gadgeteer.Modules.GHIElectronics.DisplayTE35 _lcd;

        /// <summary>
        /// The constructor of the class, he's private because the class use the design pattern Singleton
        /// </summary>
        private LCD()
        {
            _lcd = new GTM.GHIElectronics.DisplayTE35(14, 13, 12, 10);

            _lcdWidth = _lcd.Width;
            _lcdHeight = _lcd.Height;
            _lcd.BacklightEnabled = true;
            _lcd.SimpleGraphics.BackgroundColor = Gadgeteer.Color.White;
        }

        /// <summary>
        /// Getter fot the instance of the class
        /// </summary>
        public static LCD Instance
        {
            get { return _instance; }
        }

        /// <summary>
        /// Getter for the lcd width
        /// </summary>
```

```
68          public int LcdWidth
69          {
70              get { return _lcdWidth; }
71          }
72
73          /// <summary>
74          /// Getter for the lcd height
75          /// </summary>
76          public int LcdHeight
77          {
78              get { return _lcdHeight; }
79          }
80
81          /// <summary>
82          /// Method that allow access to the class
83          /// </summary>
84          /// <returns>Instance of the class LCD</returns>
85          public static LCD GetInstance()
86          {
87              if (_instance == null)
88              {
89                  _instance = new LCD();
90              }
91              return Instance;
92          }
93
94          /// <summary>
95          /// This method allow to write on the TE35 Display
96          /// </summary>
97          /// <param name="pbColor">The color of the text ↩
                (Gadgeteer.Color)</param>
98          /// <param name="pbText">The text that we want to write</param>
99          /// <param name="pbPositionX">The position X on the screen ↩
                (Default = 10)</param>
100         /// <param name="pbPositionY">The position Y on the screen ↩
                (Default = 10)</param>
101         public void DisplayText(Gadgeteer.Color pbColor, string pbText ↩
                = "", int pbPositionX = 10, int pbPositionY = 10)
102         {
103             _lcd.SimpleGraphics.DisplayTextInRectangle(pbText, ↩
                    pbPositionX, pbPositionY, _lcdWidth, _lcdHeight, ↩
                    pbColor, Resources.GetFont(Resources.FontResources.NinaB));
104         }
105
106         /// <summary>
107         /// This method clear the lcd but without a redraw
108         /// </summary>
109         public void Clear()
110         {
111             _lcd.SimpleGraphics.ClearNoRedraw();
112         }
113     }
114 }
```

Listing 1.3 – ./RFIDPiggyBank/RFIDPiggyBank/LCD.cs

1.2.3 LCDTextFields.cs

```
/*
 * Author    : Küenzi Jean-Daniel
 * Date      : 22.05.2018
 * Desc.     : This class manages LCD fields
 * Version   : 1.0.0
 */
using System;
using Microsoft.SPOT;

namespace RFIDPiggyBank
{
    public static class LCDTextFields
    {
        /// <summary>
        /// The content of the fields
        /// </summary>
        private static string _content;

        /// <summary>
        /// If we need to refresh the LCD
        /// </summary>
        private static bool _shouldBeRefresh;

        /// <summary>
        /// The position of the cursor
        /// </summary>
        private static int _cursorPosition;

        /// <summary>
        /// Getter and Setter
        /// </summary>
        public static string Content
        {
            get { return _content; }
            set { _content = value; }
        }

        /// <summary>
        /// Getter and Setter
        /// </summary>
        public static bool ShouldBeRefresh
        {
            get { return _shouldBeRefresh; }
            set { _shouldBeRefresh = value; }
        }

        /// <summary>
        /// Getter and Setter
        /// </summary>
        public static int CursorPosition
        {
            get { return _cursorPosition; }
            set { _cursorPosition = value; }
        }
    }
}
```

Listing 1.4 – ./RFIDPiggyBank/RFIDPiggyBank/LCDTextFields.cs

1.2.4 ListOfCards.cs

```
/*
 * Author    : Küenzi Jean-Daniel
 * Date      : 09.05.2018
 * Desc.     : Class that contains the list of cards (Badge) and serves to manage them
 * Version   : 1.0.0
 */
using System;
using System.Collections;
using Microsoft.SPOT;

namespace RFIDPiggyBank
{
    [Serializable]
    public class ListOfCards
    {
        /// <summary>
        /// The list of cards
        /// </summary>
        private ArrayList _cardsList;

        /// <summary>
        /// The instance of the class ListOfCards
        /// </summary>
        private static ListOfCards _instance;

        /// <summary>
        /// The constructor of the class, he's private because the class use the design pattern Singleton
        /// </summary>
        private ListOfCards()
        {
            _cardsList = new ArrayList();
        }

        /// <summary>
        /// Getter and setter for the ArrayList that handle the cards
        /// </summary>
        public ArrayList CardsList
        {
            get { return _cardsList; }
            set { _cardsList = value; }
        }

        /// <summary>
        /// Getter for the instance of the class
        /// </summary>
        public static ListOfCards Instance
        {
            get { return _instance; }
        }

        /// <summary>
        /// Method that allow access to the class
        /// </summary>
        /// <returns>Instance of the class ListOfCards</returns>
        public static ListOfCards GetInstance()
        {
            if (_instance == null)
            {
                _instance = new ListOfCards();
            }
            return Instance;
        }

        /// <summary>
        /// This method create an new object Card and add it into the ArrayList
        /// </summary>
```

```
67          /// <param name="pbName">The name of the card</param>
68          /// <param name="pbUid">The uid (RFID) of the card</param>
69          public void AddCardToList(string pbName, string pbUid)
70          {
71              pbName = (pbName == Card.DEFAULT_NAME) ? pbName + ↩
                    CardsList.Count : pbName;
72              Card card = new Card(pbName, pbUid);
73              CardsList.Add(card);
74          }
75
76          /// <summary>
77          /// This method delete an element of the ArrayList at a point ↩
                given (index)
78          /// </summary>
79          /// <param name="pbIndex">The index where we want to ↩
                delete</param>
80          public void DeleteCardFromList(int pbIndex)
81          {
82              CardsList.RemoveAt(pbIndex);
83          }
84
85          /// <summary>
86          /// This method is used to find out if the card is in the list
87          /// </summary>
88          /// <param name="pbUid">The uid of the card that we want to ↩
                search for</param>
89          /// <returns>true if we find the card | else false</returns>
90          public bool FindCardInlist(string pbUid)
91          {
92              bool result = false;
93
94              foreach (Card card in CardsList)
95              {
96                  if (card.Uid == pbUid)
97                  {
98                      result = true;
99                      break;
100                 }
101             }
102             return result;
103         }
104
105         /// <summary>
106         /// This method allow to know if the ArrayList is empty
107         /// </summary>
108         /// <returns>true if he's empty | else false</returns>
109         public bool IsEmpty()
110         {
111             bool result = (CardsList.Count > 0) ? false : true;
112             return result;
113         }
114     }
115 }
```

Listing 1.5 – ./RFIDPiggyBank/RFIDPiggyBank/ListOfCards.cs

1.2.5 RFIDReader.cs

```
/*
 * Author    : Küenzi Jean-Daniel
 * Date      : 09.05.2018
 * Desc.     : Class thaht handles the RFID module from GHI
 * Version   : 1.0.0
 */
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHI.Pins;
using Gadgeteer;
using GTM = Gadgeteer.Modules;

namespace RFIDPiggyBank
{
    public class RFIDReader
    {
        /// <summary>
        /// The instance of the class RFIDReader (Not the RFID Reader module from GHI)
        /// </summary>
        private static RFIDReader _instance;

        /// <summary>
        /// The RFID Reader module using socket 8 of the mainboard
        /// </summary>
        private Gadgeteer.Modules.GHIElectronics.RFIDReader _rfidReader;

        /// <summary>
        /// Has a badge been scanned
        /// </summary>
        private bool _isBadgeScan;

        /// <summary>
        /// The Uid of the badge that was scanned
        /// </summary>
        private string _currentUid;

        /// <summary>
        /// The constructor of the class, he's private because the class use the design pattern Singleton
        /// </summary>
        private RFIDReader()
        {
            _rfidReader = new GTM.GHIElectronics.RFIDReader(8);
            _rfidReader.IdReceived += _rfidReader_IdReceived;
            _rfidReader.MalformedIdReceived += _rfidReader_MalformedIdReceived;
            _isBadgeScan = false;
            _currentUid = "";
        }

        /// <summary>
        /// Getter for the instance of the class
        /// </summary>
        public static RFIDReader Istance
        {
            get { return _instance; }
        }

        /// <summary>
        /// Getter and Setter for the current uid
        /// </summary>
        public string CurrentUid
        {
            get { return _currentUid; }
            set { _currentUid = value; }
        }

```

```
67          /// <summary>
68          /// Getter and Setter
69          /// </summary>
70          public bool IsBadgeScan
71          {
72              get { return _isBadgeScan; }
73              set { _isBadgeScan = value; }
74          }
75
76          /// <summary>
77          /// Method that allow access to the class
78          /// </summary>
79          /// <returns>Instance of the class RFIDReader</returns>
80          public static RFIDReader GetInstance()
81          {
82              if (_instance == null)
83              {
84                  _instance = new RFIDReader();
85              }
86              return _instance;
87          }
88
89          /// <summary>
90          /// This method is if a badge is wrong scanned
91          /// </summary>
92          /// <param name="sender"></param>
93          /// <param name="e"></param>
94          private void ↩
                _rfidReader_MalformedIdReceived(GTM.GHIElectronics.RFIDReader ↩
                sender, EventArgs e)
95          {
96              Debug.Print("Badge mal scanné");
97              _currentUid = "";
98              _isBadgeScan = false;
99          }
100
101         /// <summary>
102         /// This method is when a badge is correctly scanned
103         /// </summary>
104         /// <param name="sender"></param>
105         /// <param name="id"></param>
106         private void ↩
                _rfidReader_IdReceived(GTM.GHIElectronics.RFIDReader sender, ↩
                string id)
107         {
108             Debug.Print("Uid : " + id);
109             _currentUid = id;
110             _isBadgeScan = true;
111         }
112     }
113 }
```

Listing 1.6 – ./RFIDPiggyBank/RFIDPiggyBank/RFIDReader.cs

1.2.6 SDCard.cs

```
/*
 * Author    : Küenzi Jean-Daniel
 * Date      : 09.05.2018
 * Desc.     : Class that handles the writing and reading of Cards (Badge) on the GHI SD card (SDCard module)
 * Version   : 1.0.0
 */
using System;
using System.IO;
using System.Collections;
using System.Threading;

using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT.IO;

using GHI.IO;
using GHI.IO.Storage;
using System.Text;

using GTM = Gadgeteer.Modules;

namespace RFIDPiggyBank
{
    public class SDCard
    {
        /// <summary>
        /// Name of the file where the SDCard wrote and load
        /// </summary>
        private const string FILE_NAME = "Cards.xml";

        /// <summary>
        /// The SD Card module using socket 5 of the mainboard
        /// </summary>
        private GTM.GHIElectronics.SDCard _sdCard;

        /// <summary>
        /// The instance of the class SDCard (Not the SDCard module from GHI)
        /// </summary>
        private static SDCard _instance;

        /// <summary>
        /// The constructor of the class, he's private because the class use the design pattern Singleton
        /// </summary>
        private SDCard()
        {
            _sdCard = new GTM.GHIElectronics.SDCard(5);
        }

        /// <summary>
        /// Getter for the instance of the class
        /// </summary>
        public static SDCard Instance
        {
            get { return _instance; }
        }

        /// <summary>
        /// Method that allow access to the class
        /// </summary>
        /// <returns>Instance of the class SDCard</returns>
        public static SDCard GetInstance()
        {
            if (_instance == null)
            {
                _instance = new SDCard();
            }
```

```
67              return Instance;
68          }
69
70          /// <summary>
71          /// This method serialize an ArrayList to a byte[] and save it ↩
                in a file
72          /// </summary>
73          /// <param name="pbList">The ArrayList we want to save</param>
74          public void SaveCards(ArrayList pbList)
75          {
76              if (!_sdCard.IsCardMounted) // If the SDCard isn't mounted
77              {
78                  _sdCard.Mount(); // Mount the file system
79              }
80
81              do
82              {
83                  Debug.Print("Veuillez attendre que la carte soit montée");
84              } while (!_sdCard.IsCardMounted); // We wait that the SD ↩
                    card is correctly mounted
85
86              try
87              {
88                  string sdPath = ↩
                        VolumeInfo.GetVolumes()[0].RootDirectory; // Get the ↩
                        path to the storage
89
90                  FileStream writer = new FileStream(sdPath + @"\" + ↩
                        FILE_NAME, FileMode.Create, FileAccess.Write);
91
92                  if (pbList is ArrayList)
93                  {
94                      byte[] SerializedData = ↩
                            Reflection.Serialize(pbList, typeof(ArrayList));
95                      writer.Write(SerializedData, 0, ↩
                            SerializedData.Length);
96                  }
97
98                  writer.Close();
99              }
100             catch (Exception e)
101             {
102                 Debug.Print(e.ToString());
103             }
104
105             if (_sdCard.IsCardMounted) // If the card is mounted
106             {
107                 _sdCard.Unmount(); // Unmount the card
108             }
109         }
110
111         /// <summary>
112         /// This method load the file where the byte[] is wrote and ↩
                deserialize it to an ArrayList
113         /// </summary>
114         /// <returns>An ArrayList</returns>
115         public ArrayList LoadCards()
116         {
117             if (!_sdCard.IsCardMounted) // If the SDCard isn't mounted
118             {
119                 _sdCard.Mount(); // Mount the file system
120             }
121
122             do
123             {
124                 Debug.Print("Veuillez attendre que la carte soit montée");
125             } while (!_sdCard.IsCardMounted); // We wait that the SD ↩
                    card is correctly mounted
126
127             ArrayList list = null;
128
129             try
```

```
130             {
131                 string rootDirectory = ↩
                        VolumeInfo.GetVolumes()[0].RootDirectory;
132
133                 FileStream reader = new FileStream(rootDirectory + @"\" ↩
                        + FILE_NAME, FileMode.Open, FileAccess.Read);
134
135                 byte[] SerializedData = new byte[reader.Length];
136                 reader.Read(SerializedData, 0, SerializedData.Length);
137
138                 list = ↩
                        (ArrayList)Reflection.Deserialize(SerializedData, ↩
                        typeof(ArrayList));
139
140                 if (_sdCard.IsCardMounted) // If the card is mounted
141                 {
142                     _sdCard.Unmount(); // Unmount the card
143                 }
144             }
145             catch (Exception e)
146             {
147                 list = new ArrayList();
148                 Debug.Print(e.ToString());
149             }
150
151             return list;
152         }
153     }
154 }
```

Listing 1.7 – ./RFIDPiggyBank/RFIDPiggyBank/SDCard.cs

1.2.7 ServoMotor.cs

```
1  /*
2   * Author    : Küenzi Jean-Daniel
3   * Date      : 09.05.2018
4   * Desc.     : Class that handles the servo motor from Makeblock
5   * Version   : 1.0.0
6   */
7  using System;
8  using GHI.Pins;
9  using Microsoft.SPOT;
10 using Microsoft.SPOT.Hardware;
11 using Gadgeteer;
12 using GHIElectronics.Gadgeteer;
13
14 namespace RFIDPiggyBank
15 {
16     public class ServoMotor
17     {
18         /// <summary>
19         /// The instance of the class ServoMotor
20         /// </summary>
21         private static ServoMotor _instance;
22
23         /// <summary>
24         /// The lock position for MY servo, it's not the same for all
25         /// </summary>
26         private const uint LOCK_POSITION = 1500;
27
28         /// <summary>
29         /// The unlock position for MY servo, it's not the same for all
30         /// </summary>
31         private const uint UNLOCK_POSITION = 500;
32
33         /// <summary>
34         /// This is the object PWM who controll the servo
35         /// </summary>
36         private PWM _servo;
37
38         /// <summary>
39         /// The constructor of the class, he's private because the ↩
               class use the design pattern Singleton
40         /// </summary>
41         private ServoMotor()
42         {
43             _servo = new PWM(GHI.Pins.FEZSpider.Socket11.Pwm8, 2000, ↩
                   LOCK_POSITION, PWM.ScaleFactor.Microseconds, false);
44         }
45
46         /// <summary>
47         /// Getter for the instance of the class
48         /// </summary>
49         public static ServoMotor Instance
50         {
51             get {return _instance;}
52         }
53
54         /// <summary>
55         /// Method that allow access to the class
56         /// </summary>
57         /// <returns>Instance of the class ServoMotor</returns>
58         public static ServoMotor GetInstance()
59         {
60             if (_instance == null)
61             {
62                 _instance = new ServoMotor();
63             }
64             return Instance;
65         }
66
67         /// <summary>
```

```
68          /// This method unlock the servo when call
69          /// </summary>
70          public void Unlock()
71          {
72              _servo.Duration = UNLOCK_POSITION;
73          }
74
75          /// <summary>
76          /// This method lock the servo when call
77          /// </summary>
78          public void Lock()
79          {
80              _servo.Duration = LOCK_POSITION;
81          }
82      }
83  }
```

Listing 1.8 – ./RFIDPiggyBank/RFIDPiggyBank/ServoMotor.cs