```
/*-------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomeServer is a server TCP. It's the m
 *      ain program, where all command pass before
 *      to be reply to the good client.
\*-------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Reflection;
using System.Text;
using System.Threading;

namespace RaspiHomeServer
{
    public class Server
    {
        #region Fields
        #region Constants
        // Default port communication
        private const int DEFAULT_PORT = 54565;
        // Name of the actual computer (where the server is started)
        private const string HOST_NAME = "DESKTOP-UL17MK6";
        #endregion

        #region Variables
        private List<RaspberryClient> _rpiClients;
        private CommandFilter _cmdFilter = new CommandFilter();

        private TcpListener _listener;
        private List<TcpClient> _clients;
        private TcpClient _clientRequest;
        private Dictionary<string, Dictionary<RaspberryClient, TcpClient>>
          _names;
        public readonly string _roomName;
        public readonly int _port;
        public readonly int _bufferSize = 2048;  // 2048 byte
        private bool _isRunning = false;

        private Queue<string> _messageQueue;
        #endregion
        #endregion

        #region Properties
        public List<RaspberryClient> RpiClients
        {
            get
            {
```

```
56                  return _rpiClients;
57              }
58
59              set
60              {
61                  _rpiClients = value;
62              }
63          }
64
65          public CommandFilter CmdFilter
66          {
67              get
68              {
69                  return _cmdFilter;
70              }
71
72              set
73              {
74                  _cmdFilter = value;
75              }
76          }
77
78          public TcpListener Listener
79          {
80              get
81              {
82                  return _listener;
83              }
84
85              set
86              {
87                  _listener = value;
88              }
89          }
90
91          public List<TcpClient> Clients
92          {
93              get
94              {
95                  return _clients;
96              }
97
98              set
99              {
100                 _clients = value;
101             }
102         }
103
104         public Dictionary<string, Dictionary<RaspberryClient, TcpClient>>
              ClientsNames
105         {
106             get
107             {
108                 return _names;
109             }
110
```

```
111             set
112             {
113                 _names = value;
114             }
115         }
116 
117         public bool IsRunning
118         {
119             get
120             {
121                 return _isRunning;
122             }
123 
124             set
125             {
126                 _isRunning = value;
127             }
128         }
129 
130         public Queue<string> MessageQueue
131         {
132             get
133             {
134                 return _messageQueue;
135             }
136 
137             set
138             {
139                 _messageQueue = value;
140             }
141         }
142 
143         public TcpClient ClientRequest
144         {
145             get
146             {
147                 return _clientRequest;
148             }
149 
150             set
151             {
152                 _clientRequest = value;
153             }
154         }
155         #endregion
156 
157         #region Constructor
158         /// <summary>
159         /// Constructor: Initializer
160         /// </summary>
161         public Server()
162         {
163             this.RpiClients = new List<RaspberryClient>();
164             this.Clients = new List<TcpClient>();
165             this.ClientsNames = new Dictionary<string,                    ↵
                  Dictionary<RaspberryClient, TcpClient>>();
```

```
166             this.MessageQueue = new Queue<string>();
167
168             StartListening();
169         }
170         #endregion
171
172         #region Methods
173         /// <summary>
174         /// Start the listening of the server
175         /// </summary>
176         private void StartListening()
177         {
178             // Some info
179             Console.WriteLine("Starting the {0} TCP Server on port {1}.",
                  HOST_NAME, DEFAULT_PORT);
180             Console.WriteLine();
181
182             IPAddress ipAddress = GetIPAdress();
183             this.Listener = new TcpListener(GetIPAdress(), DEFAULT_PORT);
184             IPEndPoint localEndPoint = new IPEndPoint(ipAddress,
                  DEFAULT_PORT);
185
186             this.Listener.Start();
187             this.IsRunning = true;
188
189             while (this.IsRunning)
190             {
191                 if (this.Listener.Pending())
192                 {
193                     this.NewConnection();
194                 }
195
196                 this.CheckForDisconnects();
197                 this.CheckForNewMessages();
198
199                 // Wait before sending and clearing messages
200                 Thread.Sleep(200);
201             }
202
203             // Stop the server, and clean up any connected clients
204             foreach (TcpClient v in this.Clients)
205             {
206                 this.CleanupClient(v);
207             }
208
209             this.Listener.Stop();
210         }
211
212         /// <summary>
213         /// Add new client when there is a new connection
214         /// </summary>
215         private void NewConnection()
216         {
217             bool clientIsAccepted = false;
218             // Creation of a new client at the connection
219             TcpClient newClient = this.Listener.AcceptTcpClient();
```

```
220
221             // Get the stream of the new client
222             NetworkStream netStream = newClient.GetStream();
223
224             // Modify the default buffer sizes
225             newClient.SendBufferSize = _bufferSize;
226             newClient.ReceiveBufferSize = _bufferSize;
227
228             // Print some info
229             EndPoint endPoint = newClient.Client.RemoteEndPoint;
230             Console.WriteLine();
231             Console.WriteLine("----------------------------------------");
232             Console.WriteLine("Client information");
233             Console.WriteLine("----------------------------------------");
234             Console.WriteLine("{0} Handling a new client from {1}.",
                  Environment.NewLine, endPoint);
235             Console.WriteLine();
236
237             try
238             {
239                 // Let them identify themselves
240                 byte[] messageBuffer = new byte[_bufferSize];
241                 int bytesRead = netStream.Read(messageBuffer, 0,
                    messageBuffer.Length);
242
243                 if (bytesRead > 0)
244                 {
245                     string messageRead = Encoding.UTF8.GetString
                    (messageBuffer, 0, bytesRead);
246
247                     string messageConnection = messageRead.Split('@').Last
                    ().Split(':').Last();
248
249                     try
250                     {
251                         // Get the name of the client
252                         string name = messageRead.Split('@')[1];
253
254                         if ((name != string.Empty))
255                         {
256                             // Add the player
257                             clientIsAccepted = true;
258                             this.ClientsNames.Add(name, new
                        Dictionary<RaspberryClient, TcpClient>()
                        { { this.InitializeNewRaspberryClient(messageConnection),
                        newClient } });
259                             this.Clients.Add(newClient);
260
261                             Console.WriteLine(messageRead);
262                             // Tell the current players we have a new player
263                             this.MessageQueue.Enqueue(String.Format("{0} has
                        joined the server.", name));
264                             Console.WriteLine();
265                             Console.WriteLine
                        ("----------------------------------------");
266                         }
```

```
267                     }
268                     catch (Exception)
269                     {
270                         // Wasn't either a viewer or messenger, clean up anyways.
271                         Console.WriteLine("Client wasn't able to connect.", endPoint);
272                         Console.WriteLine("----------------------------------------");
273                         Console.WriteLine();
274                         CleanupClient(newClient);
275                     }
276 
277                     // Clear the client if he doesn't meet our requirements
278                     if (!clientIsAccepted)
279                         newClient.Close();
280                 }
281             }
282             catch (Exception) { }
283         }
284 
285         /// <summary>
286         /// Clean actual client
287         /// </summary>
288         /// <param name="client"></param>
289         private void CleanupClient(TcpClient client)
290         {
291             // Clean the sent TcpClient
292             client.GetStream().Close();
293             client.Close();
294         }
295 
296         /// <summary>
297         /// Check for clients if someone is disconnected
298         /// </summary>
299         private void CheckForDisconnects()
300         {
301             // For every client
302             foreach (TcpClient client in this.Clients.ToArray())
303             {
304                 if (this.IsDisconnected(client))
305                 {
306                     try
307                     {
308                         // Get info about the messenger
309                         foreach (string name in this.ClientsNames.Keys)
310                         {
311                             if (this.ClientsNames[name].ContainsValue(client))
312                             {
313                                 // Give information of the client disconnected
314                                 Console.WriteLine();
315                                 Console.WriteLine("----------------------------------------");
316                                 Console.WriteLine("Client disconnect from the server");
```

```
317                             Console.WriteLine
                    ("----------------------------------------");
318                             Console.WriteLine("Client named {0} has
                    left.", name);
319                             Console.WriteLine();
320                             foreach (var information in this.ClientsNames
                    [name].Keys)
321                             {
322                                 if (this.ClientsNames[name].ContainsKey
                    (information))
323                                 {
324                                     Console.WriteLine("Location : " +
                    information.Location);
325                                     Console.WriteLine("Ip client :" +
                    information.IpClient);
326                                     Console.Write("Component : ");
327                                     int cnt = 0; ;
328                                     foreach (var componnent in
                    information.Components)
329                                     {
330                                         if (cnt > 0)
331                                             Console.Write("            ");
332                                         cnt++;
333                                         Console.WriteLine
                    (componnent.ToString().Split('.').Last());
334                                     }
335                                     Console.WriteLine();
336                                 }
337                             }
338                             Console.WriteLine
                    ("----------------------------------------");
339
340                             this.Clients.Remove(client);  // Remove from
                    list
341                             this.ClientsNames.Remove(name);    // Remove
                    taken name
342                             this.CleanupClient(client);   // Cleanup
343                         }
344                     }
345                 }
346                 catch (Exception) { }
347             }
348         }
349     }
350
351     /// <summary>
352     /// Check if there is a nre message from clients
353     /// </summary>
354     private void CheckForNewMessages()
355     {
356         foreach (TcpClient client in this.Clients)
357         {
358             // Get the message if there is one
359             int messageLength = client.Available;
360             if (messageLength > 0)
361             {
```

```
362                     byte[] messageBuffer = new byte[messageLength];
363                     client.GetStream().Read(messageBuffer, 0,
                    messageBuffer.Length);
364
365                     // New message from the client
366                     string messageRead = String.Format(Encoding.UTF8.GetString
                    (messageBuffer));
367                     string subject = messageRead.Split('@').Last().Split
                    (':').First();
368                     string informationInReply = messageRead.Split('@').Last
                    ().Split(':').Last();
369                     Console.WriteLine();
370
371                     switch (subject)
372                     {
373                         case "Send":
374                             Console.WriteLine
                    ("----------------------------------------");
375                             Console.WriteLine("Command send and reply");
376                             Console.WriteLine
                    ("----------------------------------------");
377                             Console.WriteLine(subject);
378                             List<TcpClient> clientsToSend =
                    this.CmdFilter.ApplyFilter(informationInReply,
                    this.RpiClients, this.ClientsNames);
379                             this.ClientRequest = client;
380                             try
381                             {
382                                 foreach (TcpClient clientToSend in
                    clientsToSend)
383                                 {
384                                     this.SendMessages(clientToSend,
                    informationInReply);
385                                 }
386                             }
387                             catch (Exception)
388                             {
389                                 // Reply ERROR_MESSAGE to the client who send
                    the command
390                                 this.SendMessages(this.ClientRequest,
                    "ERROR_MESSAGE");
391                             }
392                             Console.WriteLine
                    ("----------------------------------------");
393                             break;
394
395                         case "Reply":
396                             Console.WriteLine(subject);
397                             this.SendMessages(this.ClientRequest,
                    informationInReply);
398                             Console.WriteLine
                    ("----------------------------------------");
399                             break;
400                     }
401
402                     this.MessageQueue.Enqueue(messageRead);
```

```
403                 }
404             }
405         }
406
407         /// <summary>
408         /// Disconnect client from the server when they leave
409         /// </summary>
410         /// <param name="client"></param>
411         /// <returns></returns>
412         private bool IsDisconnected(TcpClient client)
413         {
414             try
415             {
416                 Socket clientSocket = client.Client;
417                 return clientSocket.Poll(10 * 1000, SelectMode.SelectRead) &&
                      (clientSocket.Available == 0);
418             }
419             catch (SocketException)
420             {
421                 return true;
422             }
423         }
424
425         /// <summary>
426         /// Convert the message in bytes and write in the stream of the client
427         /// </summary>
428         /// <param name="message"> message to send </param>
429         public void SendMessages(TcpClient clientToSend, string message)
430         {
431             // Encode the message
432             if (this.MessageQueue.Count != 0)
433             {
434                 byte[] msgBuffer = Encoding.UTF8.GetBytes(message);
435
436                 // Delai between each messages
437                 Thread.Sleep(200);
438                 clientToSend.GetStream().Write(msgBuffer, 0,
                      msgBuffer.Length);
439                 Thread.Sleep(200);
440
441                 Console.WriteLine(message);
442             }
443
444             this.MessageQueue.Clear();
445         }
446
447         /// <summary>
448         /// Get the ip of the raspberry
449         /// </summary>
450         /// <returns> return the IPv4 address 192.168.1.2 </returns>
451         private IPAddress GetIPAdress()
452         {
453             IPAddress result = null;
454             List<IPAddress> listAddress = Dns.GetHostAddresses
                  (HOST_NAME).ToList();
455
```

```
456             foreach (var ip in listAddress)
457             {
458                 if (ip.AddressFamily == AddressFamily.InterNetwork)
459                     result = ip;
460             }
461             return result;
462         }
463 
464         #region Initialize new client
465         /// <summary>
466         /// Initialize a raspberry pi with the information read
467         /// </summary>
468         /// <param name="rpiInformation"> Format of the string "IPRasp=
              {0};Location={1};Component={2}" to read</param>
469         /// <returns> return the new client with all information of him </
              returns>
470         private RaspberryClient InitializeNewRaspberryClient(string
              rpiInformation)
471         {
472             // Create an array of the actual string to get information
473             string[] rpiInformations = rpiInformation.Split(';');
474             string rpiIPv4 = "";
475             string rpiLocation = "";
476             string rpiComponent = "";
477 
478             foreach (var information in rpiInformations)
479             {
480                 switch (information.Split('=').First())
481                 {
482                     // Get the first value of the array
483                     // IP of the actual Raspberry
484                     case "IPRasp":
485                         rpiIPv4 = information.Split('=').Last();
486                         break;
487                     // Get the second value of the array
488                     // Location of the actual Raspberry (where to find the
                        Raspberry)
489                     case "Location":
490                         rpiLocation = information.Split('=').Last();
491                         break;
492                     // Get the third value of the array
493                     // Component of the actual Raspberry (what's the component
                         used by the Raspberry)
494                     case "Component":
495                         rpiComponent = information.Split('=').Last();
496                         break;
497                 }
498             }
499             Console.WriteLine(rpiIPv4);
500             Console.WriteLine(rpiLocation);
501 
502             // Create a client with IPv4, Default port, localisation and new
                  list of component
503             RaspberryClient client = new RaspberryClient(rpiIPv4,
                  DEFAULT_PORT, rpiLocation, new List<Component>());
504 
```

```
505             try
506             {
507                 // Get all components of the Raspberry
508                 string[] components = rpiComponent.Split('=').Last().Split
                      (',');
509 
510                 // Get all class types in the project
511                 Type[] types = Assembly.GetExecutingAssembly().GetTypes();
512 
513                 // Check all components
514                 foreach (var component in components)
515                 {
516                     // Check all class types
517                     foreach (var type in types)
518                     {
519                         if (type.Name == component)
520                         {
521                             // Instance a new component with the type result
522                             client.Components.Add((Component)
                        Activator.CreateInstance(type));
523                             break;
524                         }
525                     }
526 
527                     Console.WriteLine(component);
528                 }
529             }
530             catch (Exception e)  // Message unreadable by the program
531             {
532                 Console.WriteLine(e);
533             }
534 
535             // Add final client to the main list
536             this.RpiClients.Add(client);
537             Console.WriteLine();
538 
539             return client;
540         }
541         #endregion
542         #endregion
543     }
544 }
545 
```