```
/*--------------------------------------------------*\
 * 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.Globalization;
using System.Linq;
using System.Net.Sockets;
using System.Reflection;
using System.Text;

namespace RaspiHomeServer
{
    public class CommandFilter
    {
        #region Variables
        private RaspiHomeCommands _rhCommands;
        private string _sentence = "";
        #endregion

        #region Properties
        public RaspiHomeCommands RhCommands
        {
            get
            {
                return _rhCommands;
            }

            set
            {
                _rhCommands = value;
            }
        }

        public string Sentence
        {
            get
            {
                return _sentence;
            }

            set
            {
                _sentence = value;
            }
        }
        #endregion
```

```csharp

        #region Constructor
        /// <summary>
        /// Constructor: Initializer
        /// </summary>
        public CommandFilter()
        {
            this.RhCommands = new RaspiHomeCommands();
        }
        #endregion

        #region Methods
        /// <summary>
        /// Receive the order and treat to find raspberrys with the order
        /// </summary>
        /// <param name="paramSentence"> Sentence in entrence </param>
        /// <param name="paramRaspberryClients"> List of clients informations </param>
        /// <param name="paramClientsName"> Dictionnary of every clients registered </param>
        /// <returns></returns>
        public List<TcpClient> ApplyFilter(string paramSentence, List<RaspberryClient> paramRaspberryClients, Dictionary<string, Dictionary<RaspberryClient, TcpClient>> paramClientsName)
        {
            try
            {
                // Remove characters
                this.Sentence = RemoveDiacritics(paramSentence);

                List<TcpClient> result = new List<TcpClient>();

                string action = "";
                string location = "";
                string componentWithoutAction = "";

                // Get the component word in the sentence
                string component = this.GetComponentFromSentence(this.Sentence);

                Type componentType = null;
                string actionValue = "";

                // Different usage of the order between an component with action and without one
                // Writes values and send to the client the information or read values
                if (component != "")
                {
                    // Get the action word with in the sentence
                    action = this.GetActionFromSentence(this.Sentence);
                    // Get the property name with the action word
                    actionValue = ReadValueOfSelectedComponent(action);
                    // Get the class type founded with the component name
                    componentType = this.GetComponentType(component);
                }
                else
                {
                    // Get the sensor component word in the sentence
                    componentWithoutAction = GetIndependantComponentFromSentence(this.Sentence);
                    // Get the class type founded with the component name
                    componentType = this.GetComponentType(componentWithoutAction);
                }

                // Check every clients
                foreach (var rpiClient in paramRaspberryClients)
                {
                    // Get the location word in the sentence
                    location = this.GetSentenceLocationOrRaspberryLocation(this.Sentence, rpiClient);

                    // Check every clients at this location
                    if (rpiClient.Location.ToLower() == location.ToLower())
                    {
                        // Check every clients at this location with this component
                        foreach (var itemType in rpiClient.Components)
                        {
                            // Check if the type is the same
                            if (itemType.GetType() == componentType)
                            {
                                if (action != "")
                                {
                                    // Write the new value string informations
                                    this.WriteValue(itemType, action, itemType.GetType().GetProperty(actionValue));
                                    foreach (var name in paramClientsName.Keys)
                                    {
                                        if (paramClientsName[name].ContainsKey(rpiClient))
                                        {
                                            // Add the TCPClient inside the dictionnay
                                            result.Add(paramClientsName[name][rpiClient]);
                                        }
                                    }
                                }
                                else
                                {
                                    foreach (var name in paramClientsName.Keys)
                                    {
                                        if (paramClientsName[name].ContainsKey(rpiClient))
                                        {
                                            // Add the TCPClient inside the dictionnay
                                            result.Add(paramClientsName[name]
                    [rpiClient]);
                                        }
                                    }
                                }
                            }
                        }
                    }
                }

                return result;
            }
            catch (Exception ex)
            {
                string errorCommandFilter = ex.Message;
                return null;
            }
        }

        /// <summary>
        /// Find location if exist, else all location
        /// </summary>
        /// <param name="sentence"></param>
        /// <returns></returns>
        private string GetSentenceLocationOrRaspberryLocation(string sentence, ↩
           RaspberryClient rpiClient)
        {
            string result = "";
            string[] words = sentence.ToLower().Split(' ');

            foreach (var word in words)
            {
                if (this._rhCommands.RaspiHomeLocationKnown.Contains(word))
                {
                    result = word;
                    break;
                }
            }

            if (result == "" || result == "maison")
                result = rpiClient.Location;

            return result;
        }

        /// <summary>
        /// Get the componnent called
        /// </summary>
        /// <param name="sentence"></param>
        /// <returns></returns>
        private string GetComponentFromSentence(string sentence)
        {
            string result = "";
            string[] words = sentence.ToLower().Split(' ');

            foreach (var word in words)
            {
```

```csharp
                if (this._rhCommands.RaspiHomeComponentKnown.Contains(word))
                {
                    result = word;
                    break;
                }
            }

            return result;
        }

        /// <summary>
        /// Get the componnent called without special component connected to a special action
        /// </summary>
        /// <param name="sentence"></param>
        /// <returns></returns>
        private string GetIndependantComponentFromSentence(string sentence)
        {
            string result = "";
            string[] words = sentence.ToLower().Split(' ');

            foreach (var word in words)
            {
                if (this._rhCommands.RaspiHomeComponentWithoutActionKnown.Contains(word))
                {
                    result = word;
                    break;
                }
            }

            return result;
        }

        /// <summary>
        /// Find location exist
        /// </summary>
        /// <param name="action"></param>
        /// <returns> the action linked to the action word </returns>
        private string GetActionFromSentence(string sentence)
        {
            string result = "";
            string[] words = sentence.ToLower().Split(' ');

            foreach (var word in words)
            {
                if (this._rhCommands.RaspiHomeActionKnown.Contains(word))
                {
                    result = word;
                    break;
                }
            }

            return result;
        }

        /// <summary>
        /// Find all client who have the object in the sentence
        /// </summary>
        /// <param name="componentName"></param>
        /// <returns>the object type</returns>
        private Type GetComponentType(string componentName)
        {
            Type result = null;
            Type[] types = Assembly.GetExecutingAssembly().GetTypes();

            foreach (var typeOfComonent in types)
            {
                if (typeOfComonent.Name == this._rhCommands.RaspiLanguageTranslation[componentName])
                {
                    result = typeOfComonent;
                    break;
                }
            }

            return result;
        }

        /// <summary>
        /// Read properties value of classes
        /// </summary>
        /// <param name="actionName"> name used to change the good property </param>
        /// <returns> return the name of the property to change the value </returns>
        private string ReadValueOfSelectedComponent(string actionName)
        {
            string result = "";

            foreach (var actionKeys in this._rhCommands.RaspiBooleanCommandTranslation.Keys)
                if (actionKeys == actionName)
                {
                    // Find the Value of the dictionary trough the inner dictionary to get the first value
                    result = this.RhCommands.RaspiBooleanCommandTranslation[actionName].First().Key;
                    break;
                }

            return result;
        }

        /// <summary>
        /// Search the val to change
        /// </summary>
        /// <param name="component"></param>
        /// <param name="action"></param>
        /// <param name="typeVariable"></param>
        private void WriteValue(Component component, string action,
            PropertyInfo typeVariable)
        {
            switch (typeVariable.PropertyType.Name)
            {
                case "Boolean":
                    // Set the new value dynamicaly with value registered in an boolean dictionary
                    typeVariable.SetValue(component, this._rhCommands.RaspiBooleanCommandTranslation[action][typeVariable.Name]);
                    break;
                case "Double":
                    break;
                case "Int16":
                case "Int32":
                case "Int64":
                    break;
            }
        }



        /// <summary>
        /// Stack Overflow solution to delete accents in strings
        /// http://stackoverflow.com/questions/249087/how-do-i-remove-diacritics-accents-from-a-string-in-net
        /// </summary>
        /// <param name="str"></param>
        /// <returns></returns>
        static string RemoveDiacritics(string sentence)
        {
            var normalizedString = sentence.Normalize(NormalizationForm.FormD);
            var stringBuilder = new StringBuilder();

            foreach (var c in normalizedString)
            {
                var unicodeCategory = CharUnicodeInfo.GetUnicodeCategory(c);
                if (unicodeCategory != UnicodeCategory.NonSpacingMark)
                {
                    stringBuilder.Append(c);
                }
            }

            return stringBuilder.ToString().Normalize(NormalizationForm.FormC);
        }
        #endregion
    }
}
```