```
/*
 * Calculateur_Li
 *
 * Une classe pour réaliser le calcul de la valeur Li en
 * utilisant la sensibilité et le gain du capteur Electret
 * MAX4466.
 *
 * Voir les notes de cours "Conception des objets (IIB)" pour les
 * calculs à effectuer.
 *
 * Note: Cette classe contient un objet de classe
 *       Calculateur_VRMS pour calculer la valeur dBV du signal
 *       échantillonné.
 *
 * Convention:
 *  Variables -> camelCase
 *  Classes, fonctions -> PascalCase
 *  Constantes, types utilisateurs -> SNAKE_CASE
 *
 * GPA788 - ETS
 * T. Wong
 * 09-2018
 * 08-2020
 */
#ifndef CALCULATEUR_LEQ_H
#define CALCULATEUR_LEQ_H

// Pour pouvoir utiliser un objet de type Calculateur_VRMS
#include "calculateur_li.h"

class Calculateur_Leq {
public:
  // Pour le microphone Electret une application de 94 dB SPL
  // produit -44 dBV/Pa à sa sortie. Le gain du MAX4466 est par
  // défaut réglé à 125 ou 42 dBV.
  Calculateur_Leq(double Ts, uint16_t VrmSamples, uint16_t LiSamples){
    mTs = Ts;
    mVrmSamples = VrmSamples;
    mLiSamples = LiSamples;
  }
    // Empêcher l'utilisation du constructeur de copie
  Calculateur_Leq(const Calculateur_Leq& other) = delete;
  // Empêcher l'utilisation de l'opérateur d'assignation
  Calculateur_Leq& operator=(const Calculateur_Leq& other) = delete;
  // Empêcher l'utilisation du constructeur par déplacement
  Calculateur_Leq(Calculateur_Leq&& other) = delete;
  // Empêcher l'utilisation de l'opérateur de déplacement
  Calculateur_Leq& operator=(Calculateur_Leq&& other) = delete;

  ~Calculateur_Leq() = default;  // Destructeur

  /* -------------------------------------------------------------
     Accesseurs des données membres de la classe
     ------------------------------------------------------------- */
  inline double GetLeq() const { return m_Leq; }

  inline uint16_t GetNbSamples() const { return d.GetNbSamples(); }
  inline uint16_t GetTotalSamples() const { return d.GetTotalSamples(); }
  inline double GetVrms() const { return d.GetVrms(); }
```

```
60  inline double GetdBV() const { return d.GetdBV(); }
61  inline uint8_t GetAPin() const { return d.GetAPin(); }
62  inline double GetVMax() const { return d.GetVMax(); }
63  inline int16_t GetAdcMax() const { return d.GetAdcMax(); }
64  inline double GetLi() const { return d.GetLi(); }
65  inline double GetP() const { return d.GetP(); }
66  inline double GetM() const { return d.GetM(); }
67  inline double GetG() const { return d.GetG(); }
68  inline uint32_t GetTs() const { return mTs; }
69  inline uint16_t GetVrmSamples() const { return mVrmSamples; }
70  inline uint16_t GetLiSamples() const { return mLiSamples; }
71  inline uint32_t GetNbLi() const { return d.GetNbLi();}
72  inline void SetNbLiSamples(uint32_t nbLiSamples) {mLiSamples = nbLiSamples; }
73  inline void SetNbVrmsSamples(uint32_t nbVrmSample) {mVrmSamples = nbVrmSample; }
74  inline void ResetNbLi(){d.ResetNbLi();}
75
76
77
78
79  /* -------------------------------------------------------------
80     Services publics offerts
81     ------------------------------------------------------------- */
82  // Utiliser Accumulate() de l'objet de classe Calculateur_VRMS
83  // pour accumuler les valeurs du capteur sonore.
84  // Note: La temporisation est la responsabilité de l'utilisateur.
85  void Accumulate() {
86
87    waitUntil(GetTs());
88
89    d.Accumulate();
90
91
92  }
93  // Utiliser Compute() de l'objet de classe Calculateur_VRMS
94  // pour calculer la valeur rms du signal sonore et ensuite
95  // calculer Li du signal.
96  // Note: La temporisation est la responsabilité de l'utilisateur.
97  double Compute() {
98
99
100   if(GetNbSamples() == mVrmSamples){
101
102     d.Compute();
103     mSumLeq += mTs * mVrmSamples * pow(10, 0.1 * GetLi());
104   }
105
106
107   if(GetNbLi() == mLiSamples && GetTotalSamples() != 0){
108     m_Leq = 10*log10((1/(mTs * mVrmSamples * mLiSamples)*mSumLeq));
109     mSumLeq = 0;
110     ResetNbLi();
111
112     return 1;
113   }
114   else{
115
116     return 0;
117   }
118   return 0;
119 }
```

```
120
121   void waitUntil(uint32_t w) {
122   uint32_t t{millis()};
123   // Attendre w millisecondes
124   while (millis() < t + w) {}
125 }
126
127 private:
128   // Objet de classe Calculateur_VRMS pour réaliser les calculs
129   // Vrms et dBV du signal échantillonné.
130   // La relation entre la classe Calculateur_VRMS et la classe
131   // Calculateur_Li est une relation de "composition".
132   Calculateur_Li d;
133   // Pour le calcul de Li
134   double m_Leq;                     // Niveau d'énergie sonore au temps ti
135   double mSumLeq;
136   double ti;
137   double mTs;
138   uint16_t compteur;
139   uint16_t mVrmSamples;
140   uint16_t mLiSamples;
141 };
142
143 #endif
144
```