Lycée TOUCHARD WASHINGTON

Programmation Orientée Objet

TD n°3

SNir 1
2021-2022

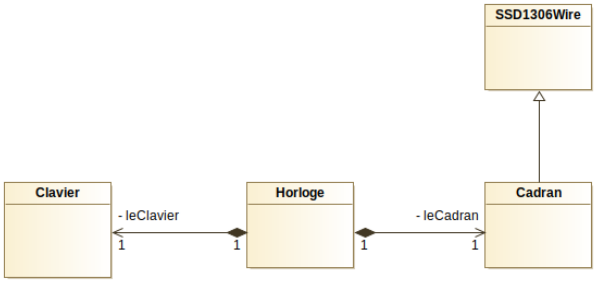
Horloge

Prérequis : La librairie pour piloter un écran SSD1306 OLED 0,96'' (128x64) avec un ESP32 doit être installée. L'écran est connecté via le bus I2C à l'ESP32.

1- Création du projet

Avec l'outil **PlatformIO**, initialisez un nouveau projet nommé **Horloge** et ouvrez-le avec votre environnement de développement.

Dans un premier temps, vous allez ajouter les différentes classes du projet. Chaque classe fera ensuite l'objet d'une mise au point individuel et permettra de répondre au diagramme de classes suivant :



- Ajoutez à votre projet une nouvelle classe nommée **Cadran**, cette classe hérite de la classe **SSD130006Wire**.
- Ajoutez également la classe **Clavier**.
- Terminer par la création de la classe **Horloge**, sans se préoccuper des réactions de compositions pour l'instant.
- Ajoutez le fichier main.cpp avec ces fonctions **setup**() et **loop**().

2- Mise au point de la classe Cadran

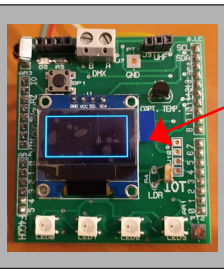
À partir des éléments d'écrit dans le cours d'introduction à la programmation-objet, compléter la classe Cadran pour être conforme à la déclaration suivante :

```
#include <Arduino.h>
#include <SSD1306Wire.h>
#include "font.h"
class Cadran : public SSD1306Wire // Relation d'héritage
{
  public:
    Cadran (const int16_t _dx=15, const int16_t _dy=16);
    void Afficher (const int _heures, const int _minutes);
    void Afficher (const String _texte,const int _valeur);
  private:
    int16_t dx; /// Coordonnée en x du 1er caractère à afficher dans le cadran
    int16_t dy; /// Coordonnée en y du 1er caractère à afficher dans le cadran
};
```

Ajoutez à votre constructeur le tracé du rectangle en bordure de l'écran OLED en plus de l'initialisation des attributs **dx** et **dy**. Complétez le codage des méthodes en tenant compte des attributs **dx** et **dy**.

Testez votre classe avec le programme suivant, que constatez-vous :

```
#include <Arduino.h
#include "cadran.h"
Cadran leCadran;
void setup()
{
}
void loop()
{
}
```



Résultats attendu

Explication du problème : Avec la relation d'héritage, l'initialisation du composant arrive trop tôt dans le processus de lancement du programme, l'ensemble des composants n'est pas encore opérationnel. Vous pouvez vérifier avec putty sur la liaison **/dev/ttyUSB0**, on voit ESP32 redémarrer de manière intempestive.

Solution : Il est nécessaire de procéder en deux temps :

- ***Déclaration d'un pointeur*** de manière globale au lieu d'une instance.
- ***Allocation dynamique*** de la classe dans la fonction **setup()**.

```
#include <Arduino.h
#include "cadran.h"
Cadran  *leCadran;
void setup()
{
    leCadran = new Cadran;
}
void loop()
{
}
```

L'opérateur **new** alloue dynamiquement la mémoire nécessaire pour créer une instance de la classe **Cadran**. Sa construction de l'instance se trouve ainsi retardé dans la fonction **setup**().

L'accès aux membres de la classe se fait désormais avec **->** au lieu du point.

Vérifier en appelant à la suite dans le **setup()** la méthode **Afficher** avec les valeurs 10 et 5 comme paramètres. Puis dans un deuxième temps avec les valeurs « HH » et 3.

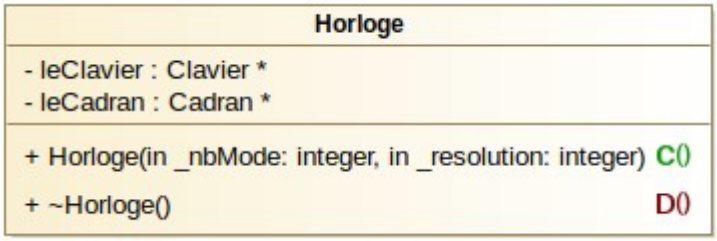
3- Classe Clavier

Reprenez la classe **Clavier** qui a permis de tester l'afficheur du TD1, effectuez les quelques adaptations nécessaires à l'application **Horloge**.

4- Codage de la classe Horloge

Pour ne pas être confronté aux mêmes problèmes rencontrés lors de la mise au point de la classe **Cadran**. Les relations de compositions doivent être implémentées de manière dynamique.

Le constructeur doit initialiser les attributs et établir la relation de composition dynamiquement avec l'opérateur **new**. Il est maintenant nécessaire d'ajouter un destructeur chargé de libérer la mémoire allouée par le constructeur. La représentation de la classe **Horloge** se trouve ainsi modifiée :



Si l'opérateur **delete** libère la mémoire obtenue avec **new**. Le destructeur est codé de la manière suivante, :

```
Horloge::~Horloge()
{
    delete leClavier;
    delete leCadran;
}
```

Réalisez ces relations de composition, vérifiez que l'initialisation se fait correctement.

Pour la gestion du temps écoulé, ajoutez un attribut nommé **tempsPrec** de type **uint64_t** qui représente la valeur du temps à l'instant précédent. Le constructeur initialise cet attribut grâce à la fonction **millis()**. Elle retourne le nombre de millisecondes écoulé depuis la mise en route du système.

Complétez le codage de la classe **Horloge** avec les éléments développés pendant le cours en respectant le diagramme de séquence et le diagramme d'états-transitions.

Dans la méthode **ActualiserHeure()**, afin de s'assurer qu'une minute c'est écoulé, un nouvel appel à la fonction **millis()** permet à nouveau de connaitre le temps écoulé depuis la mise en route du système, une simple soustraction avec l'attribut **tempsPrec** permet de déterminer la durée écoulée. Si cette durée est égale à une minute, **tempsPrec** reçoit la nouvelle valeur retournée par la fonction **millis()** et le cycle repart pour une minute. Pour la mise au point du programme, le temps d'attente pourra être réduit à une seconde.

Reste la méthode **ChangerMode()** qui n'a pas été abordée pendant l'étude en cours. Son rôle est de passer au **mode** suivant sans jamais atteindre la valeur **nbMode**. Si c'est le cas, l'attribut **mode** reprend la première valeur parmi les modes de l'horloge. L'opérateur modulo permet de répondre facilement à cette problématique.