Lycée TOUCHARD WASHINGTON

Section des Techniciens Supérieurs
Cybersécurité Informatique Electronique
Option Informatique et Réseaux

Tutoriel IHM sous Qt

Projet Alarme Domestique

3ème Séquence :
De la conception à l'Intégration

Date : septembre 2024
Version : 4.3
Référence : S3 - CentraleAlarme – Conception Integration

1. Objectif

- *Étude Algorithmique*
- *Fiche de test unitaire*
- *Relations entre classes*
  - *Composition*
  - *Principe de codage en C++*
- *Signals et Slots Qt*

2. Conditions de réalisation

- *Ce fichier contient des liens hypertextes.*
- *Ressources utilisées :*

Un PC sous Linux	*Qt-creator*

3. Conception Détaillée et Test Unitaire Classe CentraleDalarme

1. Rédigez l'algorithme de la méthode ***CentraleDalarme::FabriquerCode().*** Elle prend en paramètre d'entrée une valeur numérique sous la forme d'un octet correspondant à la touche « chiffre » enfoncée. Cette méthode complète le tableau **combinaison,** attribut de la classe **CentraleDalarme** en plaçant le chiffre reçu en fonction de l'attribut **indiceCourant** qui est utilisé comme indice du tableau. Lorsque **indiceCourant** a atteint **TAILLE_CODE**, la constante définissant la taille du tableau, les chiffres précédents sont décalés, le nouveau chiffre prend la place du dernier. Exemple : 1000 - 1200 … 1234 - 2345

Rôle : Fabrique la combinaison courante de la centrale.	
Environnement :	
En entrée :	Rien
En sortie :	Rien
En Entrée / Sortie :	combinaison[TAILLE_CODE] $_{octet}$ - indiceCourant $_{entier}$
Paramètre d'entrée :	
Paramètre de sortie :	
Paramètre d'E/S :	
Paramètre de retour :	

Schéma algorithmique :	**Lexiques** :
	Constantes : TAILLE_CODE 4
	Variables locales :
	Fonctions :

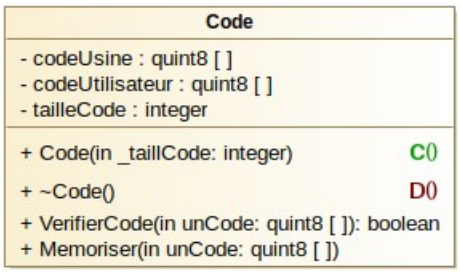
2. Proposer une fiche de test permettant de vérifier le fonctionnement de la méthode :

Référence du module testé : CentraleDalarme ::FabriquerCode		
Date du test :		
Type du test :	Fonctionnel	
Objectif du test :	Valider la fabrication de la combinaison à l'aide de la méthode **FabriquerCode**.	
Conditions du test		
État initial	Environnement du test	
L'application à son point d'entrée	Programme **centraledalarme** dans l'EDI Qt. Visualisation de la sortie d'application.	
Procédures du test L'utilisateur appuie sur les touches du clavier de la centrale, à l'aide de **qDebug**, la combinaison s'affiche au fur et à mesure de sa construction.		
Opérations		Résultats attendus
1	Le chiffre 1 est saisi sur le clavier	Combinaison = 1 0 0 0
2		
...		

3. À partir du fichier CentraleDalarme_TestUnitaire.zip, complétez le code de la méthode **CentraleDalarme::FabriquerCode()**.

4. Réalisez le compte rendu de votre test unitaire en relevant le numéro des opérations ou vous avez constaté un dysfonctionnement et la manière dont vous l'avez résolu.

4. Codage et Test de la classe Code



5. Créez un projet de type **application console** utilisant la librairie **Qt** nommé ***TestCode***. Ajoutez la classe **Code** a ce projet en respectant sa représentation UML.

Note : Les chiffres sont codés sur un octet : avec **Qt** → quint8
Ne pas oublier d'inclure <QtGlobal> pour que le types définis par **Qt** soit utilisable dans cette nouvelle classe.

6. Codez le constructeur, il alloue dynamiquement la mémoire nécessaire pour les tableaux dynamiques **codeUsine** et **codeUtilisateur** en fonction du paramètre **_tailleCode**. Dans un deuxième temps, il initialise le code usine avec la valeur 1234 et le code utilisateur avec la valeur 0000. Le code usine ne changera jamais à l'avenir. Codez également le destructeur.

7. Codez la méthode **Memoriser()**, elle remplace le code utilisateur par un nouveau code reçu en paramètre.

8. Codez la méthode **VerifierCode()**, elle retourne vrai, si la combinaison fournie correspond au code usine ou au code utilisateur, faux sinon.

9. Réalisez un programme principal permettant de valider la classe ***Code***. Pour manipuler les entrées / sorties avec l'utilisateur, vous utiliserez les flux ***cin*** et ***cout*** ainsi que les extracteurs **>>** et **<<**. Vous devez éventuellement pouvoir modifier le code utilisateur et de demander la saisie d'un code afin de vérifier s'il est valide.

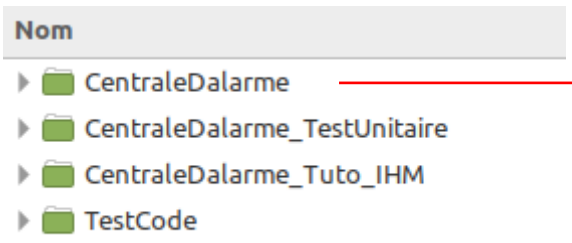
Note : ici vous pouvez supprimer la déclaration de l'instance **a** de type **QCoreApplication** et l'appel de sa méthode **exec** et remplacer le code par la proposition suivante :

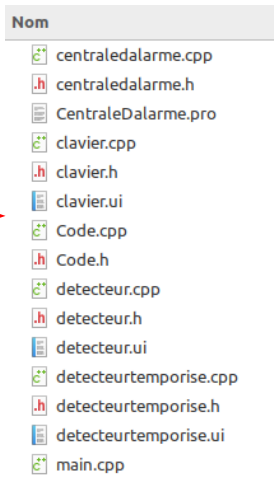
```
#include <QCoreApplication>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    return a.exec();
}
```

```
#include <iostream>
#include <QtGlobal>
#include "code.h"
using namespace std;
#define TAILLE_CODE 4
int main(int argc, char *argv[])
{
    // poursuivre le codage ici
    return 0;
}
```

5. Intégration de la classe CentraleDalarme

10. Dupliquez le répertoire **CentraleDalarme** pour obtenir un répertoire **CentraleDalarme_Tuto_IHM** que vous conserverez. Copiez dans le répertoire **CentraleDalarme** la classe de même nom ainsi que la classe **Code**.





Ajoutez à votre projet les différentes classes manquantes.

11. Complétez les différentes classes pour obtenir le diagramme de classe suivant. Dans un premier temps vous n'ajouterez pas pour l'instant la classe **Sirene** et les relations entres les classes hors mis celles qui existent déjà.

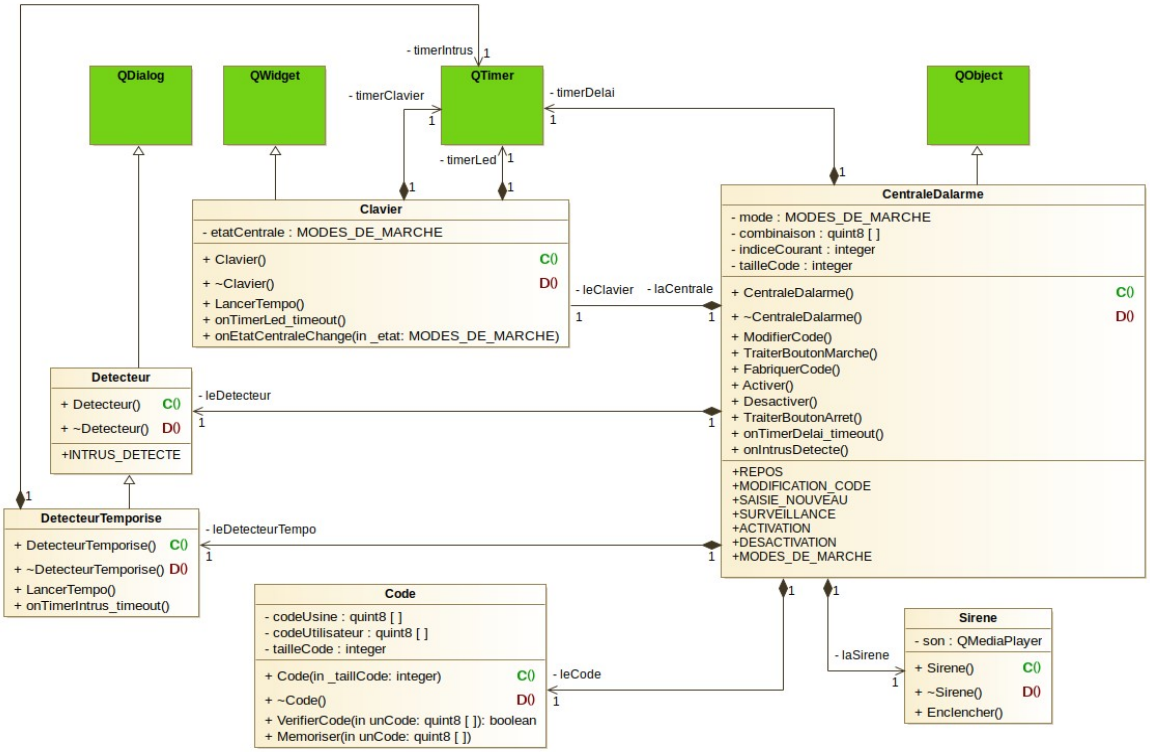


Diagramme de classes de conception

12. Le contrôle du fonctionnement de la centrale d'alarme est réparti sur les fonctions ***TraiterBoutonMarche()***, ***TraiterBoutonArrêt()*** et ***ModifierCode()*** comme le montre le diagramme des modes et états ainsi que les diagrammes de séquences présents dans le dossier d'analyse. Compléter le tableau ci-dessous en indiquant comment passer d'un état à un autre en fonction de la méthode appelée :

	État de départ	Transition	État d'arrivé
TraiterBoutonMarche (Appui sur le bouton marche pendant moins de 5 secondes)	*REPOS*	*Combinaison comporte 4 chiffres + code correcte*	*SURVEILLANCE*
TraiterBoutonArrêt (appui sur le bouton arrêt)			
ModifierCode (appui sur le bouton marche pendant plus de 5 secondes)			

À partir de ces éléments, vous ajouterez la constante énumérée **MODES_DE_MARCHE** dans la section publique de la classe **CentraleDalarme**. Prévoir également un état **ACTIVATION**, il est utilisé lorsque le propriétaire quitte les locaux surveillés avant le passage en mode surveillance… vous pouvez ajouter la gestion de cette état dans votre dossier d'analyse Modelio.

13. Pour simplifier la gestion des signaux entre les classes **CentraleDalarme** et **Clavier** afin de mettre à jour les LEDs, on propose d'ajouter le signal **Qt** nommé **EtatCentraleChange** dans la déclaration de la classe **CentraleDalarme**. Le paramètre indique l'état de la centrale.

```
signals:
    void EtatCentraleChange(ETAT_CENTRALE _nouvelEtat);
```

14. D'après le diagramme de classes de conception et le code fourni pour le test unitaire, expliquez pourquoi la relation entre la classe **CentraleDalarme** et la classe **Clavier** est-elle bidirectionnelle. Comment cela est-il implémenté ?

15. Pour le codage de la méthode **ModifierCode()**, on propose le code C++ sous **Qt** suivant :

```
void CentraleDalarme::ModifierCode()
{
    if(indiceCourant == TAILLE_CODE)
    {
        etat = MODIFICATION_CODE;
        emit EtatCentraleChange(etat);
    }
}
```

    La fonction **emit** permet d'envoyer le signal **EtatCentraleChange** avec le paramètre **etat**.

16. Pour réceptionner le signal dans la classe **Clavier**, ajoutez à cette classe un nouveau **slot public** dans la déclaration comme le montre l'exemple suivant :

```
public slots:
    void onEtatCentraleChange(CentraleDalarme::MODES_DE_MARCHE _etat);
```

    Le **slot** doit être public car il est appelé à partir d'une autre classe ici la classe **CentraleDalarme**. Ajoutez également la définition de la classe (vous compléterez sont contenu par la suite) au fichier clavier.cpp. Réalisez dans le constructeur de la classe Clavier la connexion entre le **signal** et le **slot**.

17. À partir des diagrammes de séquence et en vous inspirant de La méthode **ModifierCode()** présentée précédemment, réalisez le code C++ des méthodes **TraiterBoutonArret()** et **TraiterBoutonMarche()**.

18. Codez la méthode du slot **onEtatCentraleChange** de manière à ce que les LEDs réagissent comme prévu dans le cahier des charges. On peut prévoir un clignotement plus rapide (250 ms) lorsque la centrale est dans le mode ACTIVATION.

19. Réaliser la deuxième composition de la classe Clavier vers la classe **QTimer** pour contrôler l'appui pendant 5 secondes sur la touche MARCHE. Réalisez le codage nécessaire pour cette gestion, les signaux **pressed** et **released** (enfoncé et relâche) sont nécessaires au bon fonctionnement de cette tâche.

Avant d'aller plus loin vérifier le fonctionnement :

- Le changement de code est possible
- L'activation avec le code-usine, avec le code utilisateur
- L'arrêt avec le code-usine, avec le code utilisateur
- Le fonctionnement des LEDs est conforme au cahier des charges

6. Intégration des détecteurs

20. Réalisez les compositions avec les classes **Detecteur** et **DetecteurTemporise** comme elles apparaissent dans le diagramme de classe.

21. Ajoutez les signaux et les slots nécessaires au fonctionnement des détecteurs. Ajoutez également les méthodes **Activer()** et **Desactiver()** à la classe **CentraleDalarme**. Compléter le codage de **TraiterMarche()** et **TraiterArret()** conformément au diagramme de séquence « **Armer ou Désarmer la centrale** »

7. Codage de la classe Sirene

Pour que Qt puisse utiliser les classes Multimédia, il est nécessaire d'ajouter dans le fichier **CentraleDalarme.pro** la mention **multimedia**

```
QT   += core gui multimedia
```

On donne quelques lignes simples pour l'usage du projet :
cette 1ère ligne déclare une instance de la classe **QMediaPlayer**.

```
QMediaPlayer son;
```

Cette 2ème ligne charge le fichier en question :

```
son.setMedia(QUrl::fromLocalFile("/home/philippe/Musique/alarme.mp3"));
```

La 3ème ligne lance la lecture du fichier audio

```
son.play();
```

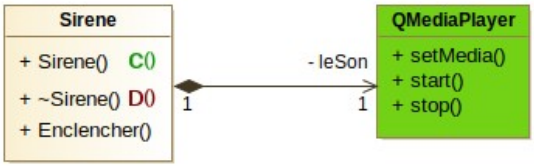
si besoin pour arrêter :

```
son.stop();
```

pour plus d'information : https://doc.qt.io/qt-6/qmediaplayer.html

22. Réalisez la classe **Sirene** conformément au diagramme de classe suivant :

    et codez les différentes méthode de la classe **Sirene**.



23. Intégrez votre classe Sirene au reste de l'application.

24. Après avoir exporter votre projet Modelio correspondant à l'analyse de la centrale d'alarme, renommez ce projet en CentraleDalarme_Conception. Ajoutez les différentes classes de **Qt** que vous avez utilisé pour votre développement dans un package nommé **Qt**. Pour chacune de ses classe vous pouvez ajoutez également le nom des méthodes et des signaux que vous avez utilisé. Reproduisez le diagramme de classes proposé à la page 5 et complétez le éventuellement. La dernière étape consiste à reprendre les diagrammes de séquence en montrant les interactions avec les classe Qt. Vous avez maintenant de quoi constituer un dossier de conception préliminaire.

Pour aller encore plus loin :

Réalisez les modifications pour la gestion de plusieurs détecteurs de différents types.

Remplacez les cases à cocher sur l'interface du clavier par des boutons que vous désactiverez (propriété enable) sans texte dessus.

Ensuite pour modifier la couleur d'un bouton, vous pouvez utiliser la méthode :

```
ui->pushButtonLedRouge->setStyleSheet(" QPushButton {background-color:red}");
```

L'appel de la méthode ui->pushButtonLedRouge->styleSheet() retourne une **QString** qui donne le style du bouton, cela permet de récupérer par exemple la texture standard d'un bouton. Visualisez avec qDebug pour la mise au point éventuellement.

Adaptez votre code pour que la gestion des LEDs soit faite par ces deux widgets.