Analyse UML et Codage C++

Projet Centrale Alarme Domestique

~ TD n°9 ~

3ème Séquence : Finalisation de l'application

Date : février 2019
Version : 4.0
Référence : TD9 – Finalisation

Plan du document

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

- *Ressources utilisées :*

Matériel	***Logiciels***
Un PC sous Linux	*La suite open office* *Qt-creator, Modelio*

3. RECOMMANDATIONS

L'ensemble des éléments obtenus, classes, opérations, attributs doivent faire l'objet d'un commentaire précis et non redondant. Le nom des variables est significatif.

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 :

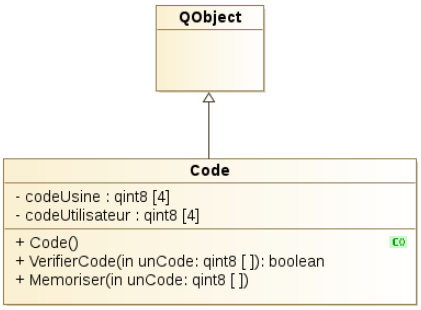
2. Proposer une fiche de test permettant de vérifier le fonctionnement de la méthode :

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.

Codage de la classe Code

1. Créez un projet de type **application console** nommé ***TestCode*** avec Qt-Creator. Ajoutez la classe code a ce projet en respectant l'extrait du diagramme de classes ci-contre :



   note : Les chiffres sont codés sur un octet : avec Qt → qint8

2. Codez le constructeur, 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.
3. Codez la méthode **Memoriser()**, elle reçoit un nouveau code sur 4 chiffres et remplace le code utilisateur.
4. Codez la méthode **VerifierCode()**, elle retourne vrai, si la combinaison fournie correspond au code-usine ou au code utilisateur, faux sinon.
5. 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 **<<**.

   note : ici vous pouvez supprimer la déclaration de l'instance **a** de type **QApplication** et l'appel de sa méthode **exec**.

6. Lorsque la classe code est opérationnelle, copiez les fichiers correspondants dans votre projet **CentraleDalarme**. N'hésitez pas à utiliser l'outil de mise au point de Qt en cas de problème.

Codage de la classe Centrale d'alarme

7. 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
8. 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 **ETAT_CENTRALE** dans la section publique de la classe **CentraleDalarme**. Prévoir un état **ACTIVATION**, il est utilisé lorsque le propriétaire quitte les locaux surveillés.

9. Pour la mise à jour des LEDs sur le clavier, on propose d'ajouter le signal **ChangementEtatCentrale** dans la déclaration de la classe **CentraleDalarme**. Le paramètre indique l'état de la centrale.

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

10. D'après le diagramme de classes ci-dessous 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 ?

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

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

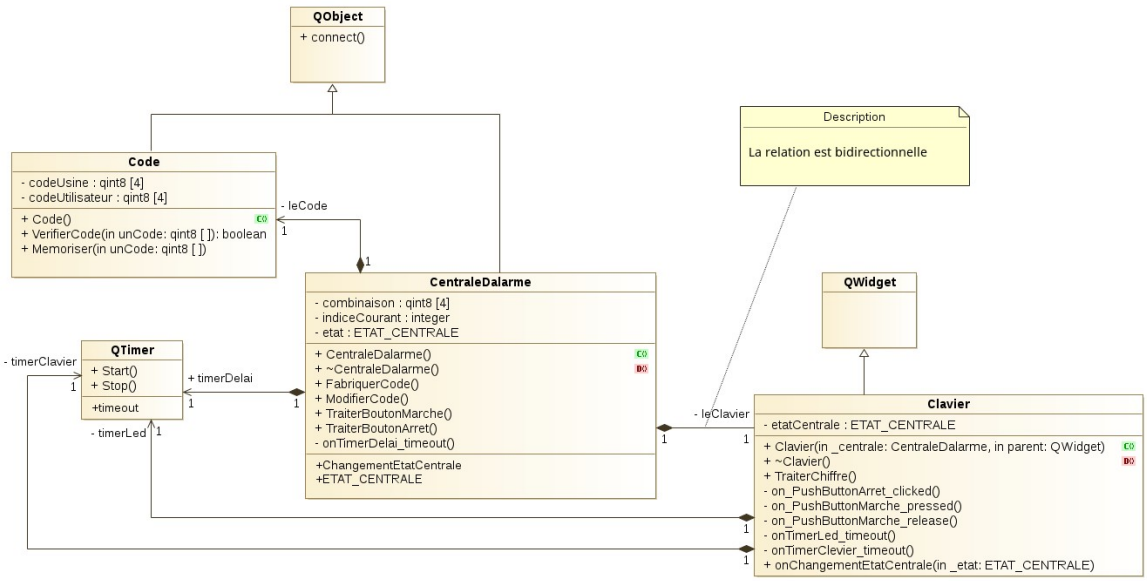
    À partir des diagrammes de séquence et en vous inspirant de ce principe, réalisez le code C++ des méthodes **TraiterBoutonArret()** et **TraiterBoutonMarche()**.

12. Dans la classe **Clavier**, ajoutez un nouveau **slot publique**, il doit être appelé depuis la classe **CentraleDalarme**, void onChangementEtatCentrale(CentraleDalarme::ETAT_CENTRALE _etat) ; Codez ensuite le corps de la méthode pour qu'en fonction de l'état de la centrale les LEDs agissent comme cela était prévu dans le cahier des charges. On peut prévoir un clignotement plus rapide (250 ms) lorsque la centrale est dans le mode ACTIVATION.

    Complétez le slot **onTimerLed_timeout() ;** pour qu'il réponde au besoin. Attention, la LED rouge ne clignote que lorsque la centrale d'alarme est dans l'état MODIFICATION_CODE.

13. Prévoir un deuxième **QTimer** pour la gestion du clavier et le slot nécessaire à sa gestion. Il va servir à attendre les 5 secondes pour la modification du code.

La figure ci-dessous reprend l'architecture du logiciel à ce stade du développement.



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

Codage de la classe Centrale d'alarme

Les détecteurs ont été étudiés lors de la séquence précédente (tutoriel IHM Qt).

14. Copiez les fichiers detecteur.ui, detecteurtemporise.h, detecteurtemporise.cpp, detecteur.cpp, detecteur.h dans votre dossier de projet et ajoutez ces fichiers existants à votre projet.
15. Sous Modelio, complétez le diagramme de classes de conception (avec les objets Qt) en faisant apparaître les détecteurs et les relations avec la classe **CentraleDarlame**.
16. Ajouter 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** »

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 un exemple :

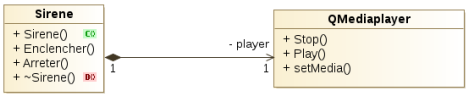
```
QMediaPlayer sirene;
sirene.setMedia(QUrl::fromLocalFile("/home/philippe/Musique/alarme.mp3"));
sirene.play();
```

si besoin pour arrêter :

```
sirene.stop();
```

pour plus d'information : http://doc.qt.io/qt-5/qmediaplayer.html

17. Réalisez la classe Sirene conformément au diagramme de classe suivant :



18. Intégrez votre classe Sirene au reste de l'application.
19. Complétez le diagramme de classe Modelio (version avec les classe Qt)
20. À partir de votre code, réalisez les diagrammes de séquence de conception utilisant les méthodes de Qt lorsque nécessaire.

Pour aller plus loin

Notre centrale d'alarme utilise plusieurs détecteurs de différents types comme l'indiquait le diagramme de classe. Réalisez les modifications nécessaires pour répondre à ce besoin.

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.

Modifiez votre code pour que la gestion des LEDs soit faite par ces nouveaux boutons.